SYSTEM'S BANK

PC88♯2

アスキー・システム・バンク（PC88♯2）

PC-8801グラフィックスのすべて

工藤丈彦・榊正憲・田村明史・横溝和宏共著

アスキー出版局

アスキー・システム・バンク

PC-8801 グラフィックスのすべて

工藤丈彦・榊 正憲
田村明史・横溝和宏 共著

PC88♯2

ASCII
SYSTEM'S BANK

Ⓐ

Ⓑ

Ⓒ

表紙の写真は，PC-8801により作成された3枚のコンピュータ・グラフィックス画面を印刷過程で合成したものです．最終的にハイ・レゾリューショナルな画を得るためにはこのような手法が最も適していると思います．

今回は体高1メートルのフラミンゴを17メートルの距離から，画角約4°，俯角10°で見下ろすという状況を設定し，すべてのシミュレーションをその状況に基づいて行っています．そのため，後に合成した場合でも不自然ではないものに仕上っているはずです．

Ⓐ，Ⓑの位置関係の違いによるパースペクティブの変化，それぞれの回転による見え方の違いなどを参考にしていただければ幸いです．

Computer Graphics Design by Shingo Nakajima
Photo by Yutaka Yagai／Studio V

Designによるロゴタイプの作成
6章リスト4

Clothによるテキスタイル・デザイン　任意の中間色や模様でブラウス，スカートなどの調和を見ることが出来る
6章リスト2

CAD（ロジック回路の作成）に挑戦
6章リスト5

数式を用いたグラフィックス（2章，3章）

ライン・スタイルによる折れ線グラフ　5章リスト1

タイル・パターンによる円グラフ　5章リスト2

メッシュによる3次元関数のプロット
3章リスト21

ダイヤモンド　2章リスト11

動物の角のシミュレーション　2章リスト17

ペイントのデモ　2章リスト19

メッシュによる3次元関数の
プロット　3章リスト20

高次関数によるプロット　3章リスト17

高次関数によるプロット
3章リスト19

指数関数によるプロット　3章リスト18

LINE命令のみによるデモ　2章リスト5

シンボル・ジェネレータとPUT文
によるシンボル・グラフ
5章リスト4

多角形によるレーダー・
チャート　5章リスト3

対数関数による両対数グラフ　5章リスト6

○コンピュータ絵本○

ワオ！　ボクは桃太郎　鬼ケ島へ行くんだい!!

ねぇ　きびだんごあげる　Wh…,Khue Khue,ピッピッ

えっほ　えっほ　やっつけた　エイ!!

ルンルン気分で太平洋

これらの画面はいずれも本書の４章中で紹介しているグラフィック・エディタによる作画です．

デザイン協力：村上　滋

田舎を背景とした時の車の色のシミュレーション　6章リスト3

パレットを変えると野山が紅葉した感じに　6章リスト3

都会を背景とした時の車の色のシミュレーション　6章リスト3

車の色の塗り変えは自由自在　6章リスト3

Gエディタのオープニング画面　4章リスト7

Gエディタによる桃太郎作画風景　4章リスト7

Roomによる6帖の部屋のレイアウト
十字カーソルで家具の位置を決めていく　6章リスト1

Roomのオープニング画面
6章リスト1

Sangoの作画風景　大きな枝を1本描き終えた
6章リスト6

見事にサンゴを描き終えた
6章リスト6

シンボル・ジェネレータでグラフィック・シンボルを作る　4章リスト5

PC-8801 グラフィックスのすべて

工藤丈彦　榊　正憲
田村明史　横溝和宏　共著

アスキー出版局

この本を読む前に

近年，著しい進展を見せるコンピュータ・グラフィックス，今やその応用分野は計算機の世界のみならず，ビジネス，コマーシャル，映画，デザイン，美術などの分野に盛んに採用され始めています．

本書は，このコンピュータ・グラフィックスを比較的低価格であるパーソナル・コンピュータで実現する手法と，そのプログラミングについて解説したものです．対象とする機種はPC-8801であり，本書中で標準システムとしているのは，本体＋カラーCRT（解像度640×200ドット）です．もしこれ以外の装置（例えば，高解像度CRT，ディスク・ユニット，漢字ROMなど）が必要とされる場合は，その都度注意書きを添えてあります．

本書は，既巻〝PC-8801 BASIC入門〟に続く第2巻として作られていますので，N_{88}-BASICのステートメント及びそのプログラミングについて，ある程度の知識があることを前提としています．従って，各プログラムで直接グラフィックスと関連のないものについては，詳しい説明は省かせて頂きました．

以下に本書の構成について簡単に紹介しておきます．

1章　グラフィックスのためのハードウェア

PC-8801を使ったグラフィックスの話に入る前に，〝コンピュータ・グラフィックスがどのようなハードウェアの元に成り立っているものなのか〟ということを，一般的なグラフィック・システムからPC-8801のグラフィック・システムへと，ハードウェアを中心に説明していきます．以降の章を理解する上で，またグラフィックスが生み出される仕組を理解する上で役立つことでしょう．

2章　基本図形を描く

グラフィックスを行う上で，直線，多角形，円などの基本図形を自由に描けるようになることは，非常に重要なことです．この章では，これらの基本図形の描き方をDDAなどの高度な手法

も含めて紹介します．

3章　任意の関数をプロットする

前章の発展として，任意の関数（2次関数や三角関数など）を自由に画面上で表現できることも，またグラフィックスとして意味があります．これらの関数はプログラミングが簡単な割には，プログラマのセンスしだいで，美しい画面を作り出すことができるのです．この章では，PC-8801グラフィックスの特徴の1つであるウィンドウを効果的に使い，代表的な関数を描いていきます．

4章　グラフィックス・ツールの紹介

この章では，以後のプログラムの内でよく使われるもの，また，読者の方々が後で利用価値の大きいものを BASIC と機械語のツールとして提供します．

5章　ビジネスへの応用

この章では少し趣を変えて，PC-8801のグラフィックスをビジネスの分野に応用してみます．内容は，各種グラフと漢字ワード・プロセッサです．これらはある程度実用に耐えるプログラムとして作られています．

6章　静的グラフィックス

ここでいう静的グラフィックスとは，次章の〝動的グラフィックス〟に対しての意味であり，アニメーションや動的ゲームなどのように動きのあるグラフィックスではないものを集めました．ロジック回路やテキスタイル・デザイン(服地のデザイン)は，これから趣味にグラフィックスを取り入れてみようとする人々には，興味深い内容でしょう．

7章　動的グラフィックス

コンピュータ・グラフィックスは確かに美しいものですが，その画像が動き出したとしたら，我々にまた新しい感動を与えるものです．この章では，動的なグラフィックスを実現する簡単な手法をいくつか紹介しながら，ボールの移動からアニメーションにまで話を広げていきます．

8章　3次元グラフィックスへの招待

2章～7章はすべて2次元平面におけるグラフィックスでしたが，この章ではさらに次元を拡張し，3次元グラフィックスを紹介します．3次元グラフィックスとは何か，またそれを実現する手段は，などについてお話しします．

以上の各章は大きく分類すると，1章～3章がコンピュータ・グラフィックスの基礎，4章～7章がその様々な分野への応用，8章が3次元グラフィックスについてです．

本書のプログラムは全章を通し，読み易さを重視してインデンテーション（段付け）を行い，必要以上のマルチ・ステートメントを避けていますから，多少長めになってしまいました．その中でも4章～7章のプログラムのいくつかは，〝実用に耐えるもの〟ということを意識して，エラー処理やヘルプ・ルーチンなどを設けていますから，かなり長いプログラムとなっています．くれぐれもタイプミスのないようにしてください．また，本書中のプログラムのほとんどは，飛び先の指定をラベル名で行っていますから，行番号について神経質になる必要はありません．

目　次

この本を読む前に……………………………………………………………………3

1章 グラフィックスのためのハードウエア………13

1.1 ———一般的なグラフィック・ディスプレイ———13
1.1.1——ラスタ・スキャン・ディスプレイ———14
1.1.2——カラー表示———15
1.1.3——マルチ・プレーン———17
1.2 ———PC-8801の画面に関するハードウェア———18
1.2.1——テキストV-RAM———18
1.2.2——グラフィックV-RAM———20
1.2.3——カラー・パレットとその他のカラー———21
1.2.4——入/出力機器———22

2章 基本図形を描く………25

2.1 ———座標系と点———26
2.1.1——座標って何———26
2.1.2——グラフィック座標系———27
2.1.3——絶対座標と相対座標———28
2.1.4——点を打つ命令PSET文———29
2.1.5——点を消す命令PRESET文———30
2.1.6——夜空 —乱数を使って点を打つ— ———30
2.1.7——LPを設定するPOINT文———31
2.2 ———直線で図形を描く———31
2.2.1——直線を引くには———31
2.2.2——直線を引く命令LINE文———37

2.2.3 ── 三角形, 四角形, 五角形 … ──────38
2.3 ───円を描く──────────45
2.3.1 ──円を描くには──────45
2.3.2 ──円を描く命令CIRCLE文──────52
2.4 ───図形に色を塗る──────────54

3章 任意の関数をプロットする………57

3.1 ───任意の関数をプロットするには──────────57
3.1.1 ──ワールド座標とウィンドウ──────57
3.1.2 ──方程式を描くには──────59
3.2 ─── 1 次関数──────────61
3.3 ─── 2 次関数（高次関数）──────────65
3.4 ───ルート関数と分数関数──────────69
3.4.1 ──ルート関数(無理関数)──────69
3.4.2 ──分数関数──────71
3.5 ───対数関数──────────72
3.6 ───指数関数──────────74
3.7 ───三角関数──────────75
3.7.1 ──y＝sinX(正弦関数)──────75
3.7.2 ──y＝cosX(余弦関数)──────77
3.7.3 ──y＝tanX(正接関数)──────78
3.7.4 ──サインとコサインを合成する──────79
3.8 ───媒介変数を用いた曲線──────────81

4章 グラフィック・ツールの紹介………87

4.1 ───ツールの必要性──────────87
4.2 ───BASICによるツール──────────87
4.2.1 ──中間色ジェネレータ(MID.SEL)──────88
4.2.2 ──タイル・パターン・ジェネレータ──────93
4.2.3 ──シンボル・ジェネレータ──────97
4.2.4 ──グラフィック・エディタ──────100

4.3 ——機械語によるツール(Glib)——115
4.3.1——Glib——115
4.3.2——CLS——118
4.3.3——ROLL——122
4.3.4——パターン反転——125
4.3.5——クロス・カーソル——127
4.3.6——漢字1行PUT——129
4.3.7——入力の仕方——132
4.3.8——ライブラリのロード——134
4.3.9——その他のグラフィックI/O——135

5章 ビジネスへの応用……139

5.1 ——グラフを描く——139
5.1.1——ウィンドウとビューポートの活用——139
5.1.2——スケールの決め方——139
5.1.3——数字の振り方——140
5.2 ——各種グラフ——142
5.2.1——折れ線グラフ——142
5.2.2——円グラフ——147
5.2.3——レーダー・チャート——151
5.2.4——シンボル・グラフ——156
5.2.5——対数グラフ——160
5.3 ——漢字への応用——164
5.3.1——漢字ワード・プロセッサ——164
5.3.2——グラフへの漢字の導入——191

6章 静的グラフィックス……193

6.1 ——白黒3ページ・モードを使った部屋のレイアウト(Room)——193
6.1.1——画面上の位置指定——193
6.1.2——面画の合成(3画面の使用)——195
6.1.3——プログラムの説明——195

6.1.4——Roomの使い方——201
6.2 ——タイリングを使った服地のシミュレーション(Cloth)——202
6.2.1——GET@とPUT@を用いた切り絵のテクニック——202
6.2.2——プログラムの説明——204
6.2.3——Clothの使い方——209
6.3 ——LINE文を主体とした作画(Car)——210
6.3.1——LINE文よる作画の長所と短所——210
6.3.2——Carの使い方——214
6.4 ——白黒パターンのデザイン(Design)——215
6.4.1——プログラムの説明——215
6.4.2——Designの使い方——223
6.5 ——ロジック回路を描く(Logic)——228
6.5.1——プログラムの説明——228
6.5.2——Logicの使い方——237
6.6 ——ポリゴンの利用(Sango)——242

7章 動的グラフィックス………247

7.1 ——パターンの移動——247
7.2 ——カラー・パレットの利用——254
7.3 ——ページングの利用——260
7.4 ——GET@／PUT@の利用——265

8章 3次元グラフィックスへの招待………277

8.1 ——3次元グラフィックスとは——277
8.2 ——図形の2次元変換——279
8.2.1——平行移動——279
8.2.2——拡大・縮小——281
8.2.3——回転——282
8.2.4——クリッピングとウィンドウ——284
8.3 ——3次元変換——288
8.3.1——平行移動——288

8.3.2——拡大・縮小——289
8.3.3——回転——289
8.3.4——視野とクリッピング——290
8.3.5——透視変換——293
8.4 —— 3次元パッケージ——293
8.5 ——ワイヤ・フレーム——297

APPENDIX Glibソース・リスト……303
索引……316

1章 グラフィックスのためのハードウェア

最近，パーソナル・コンピュータのハードウェアやソフトウェアの発達には，目を見張るものがあります．それが端的に現れているのが，〝グラフィックス〟でしょう．640×400の分解能，1ドットごとに色が指定できる，カラー・マップ・テーブル，漢字が簡単に扱える，ウィンドウやクリッピング，点線が簡単に引ける，…など，ちょっと前では簡単に実現できそうもなかった機能が，低価格のパーソナル・コンピュータでも可能になってきました．本書が対象としているPC-8801も例外なく，上で述べたような多彩なグラフィックス機能を持っています．そこで，この章では，まずこれらパーソナル・コンピュータの多彩なグラフィックス機能の土台となっている，一般のグラフィック・ディスプレイ（但し，ここでいう〝グラフィック・ディスプレイ〟とは，単なる画面だけではなくV-RAMなどのグラフィックスに必要なハードウェアを含んでいるものです．）についてお話しします．次に，PC-8801の画面に関するハードウェアについてお話しします．そのため，抽象的な部分や専門用語が数多く出てくるので，コンピュータ・グラフィックス界の実情にあまり詳しくないという方には，難しいかもしれません．しかし，この章を一度でも読んでおけば，後々の理解に役立つことと思います．

1.1 一般的なグラフィック・ディスプレイ

現在，コンピュータ・グラフィックスの入/出力装置として，多くの種類のグラフィック・ディスプレイ（表1）が利用できますが，その中で最もよく使われるのは，テレビジョン・モニタのようなCRT（Cathode-Ray Tubeの略）でしょう．このCRT上に表示された図形は，容易に消去や修正が行えるという利点がある反面，電源を切るとCRT上の図形が消えてしまうという欠点があります．このような図をソフト・コピー（soft copy）といいます．これに対して，同じグラフィック・ディスプレイでも，機械式プロッタや，プリンタなどで描かれた紙上の図形は，出力装置の電源を切っても，そのまま残っています．この図はハード・コピー（hard copy）といいます．欠点は，図形を描く速度が遅く，描かれた図形に対して容易に修正が行えないことです．

グラフィック・ディスプレイは，ソフト・コピー，ハード・コピーの違いだけではなく，その

表示方式の違いにより，

ランダム・スキャン（random scan）方式
ラスタ・スキャン　（raster scan）方式

の二種類に大別できます．ランダム・スキャン方式は，ベクタ・スキャン方式とも呼ばれ，プロッタのペンやCRTのビームが，表示図形の上だけを動いて図形を表示する方法（図1）のことです．この方式によって描かれた図形を，ベクタ・グラフィックス（vector graphics）といいます．ラスタ・スキャン方式は，表示図形のあるなしにかかわらず，家庭用のテレビと同じように，プリンタヘッドやCRTのビームが，上から下へと順次スキャン（走査）しながら，図形の存在する点を表示する方法(図2)で，この方法によって描かれた図形を，ラスタ・グラフィックス(raster graphics)といいます．いずれにしろ一長一短がありますが，ここではPC-8801の採用しているラスタ・グラフィックスについてのみ，お話しすることにして，より詳しいことやベクタ・グラフィックスのことは専門書に任せることにします．

1.1.1 ラスタ・スキャン・ディスプレイ

カラーや濃淡などの多彩な表現が可能なフレーム・バッファ・ディスプレイ(frame buffer display)

ベクタ・スキャン・ディスプレイの構造

表示例

図1　ベクタ・グラフィックス

ラスタ・スキャン・ディスプレイの構造

表示例

図2　ラスタ・グラフィックス

- 入力装置
  - キーボード
  - ライトペン
  - ジョイスティック
  - タブレットとスタイラス
  - トラック・ボール
  - マウス
  - ビデオ・カメラ
  - ⋮
- 出力装置
  - CRT
  - プリンタ
  - プロッタ
  - ⋮

表1　グラフィックスの入/出力装置

は，メモリICの値下がりと共に比較的安値になり，広く利用されるようになってきました．このディスプレイの構造はきわめて簡単で，図3のように，表示する画像の輝度情報を蓄えるフレーム・バッファと，家庭用テレビと同じような構造を持つテレビジョン・モニタと，フレーム・バッファ内の情報をモニタへ送る働きをするディスプレイ・コントローラの3つから成り立っています．

画面に表示される図形は，縦横に整然と分割された1つ1つの点から構成されていて，この最小単位である四角の部分を，画素（picture element），あるいはピクセル（pixel）と呼びます．そして，横方向のピクセルの集まりをスキャン・ライン（scan line）といい，スキャン・ライン全体をフレーム(frame)と呼びます．このフレーム全体の輝度情報を，各々のピクセルについて記憶しているのが，フレーム・バッファであるということはすでにお話ししましたが，1つのピクセルに何段階の階調を与えるかによって，1つのピクセル当りのビット数が決まります．図3のように線で表すことのできる図形では，点が表示されるかされないかの2通りでよいので，1ビットで間に合います．それが，〝0〟だと消点として，〝1〟だと輝点として表示されます．そのため，表示しようとする点に対応するフレーム・バッファのビットを〝1〟に，表示しないでいる点に対応するフレーム・バッファのビットを〝0〟にすれば，任意の図形を表示することができます．この時，フレーム・バッファには，図3に示すように，2進数がマトリックス状に蓄えられていて，それからも表示される図形が推定できます．

1.1.2　カラー表示

カラー・グラフィックスは，色が1ドットごとに指定できるかできないかによって，

フルカラー・グラフィックス

セミカラー・グラフィックス

の二種類に分けられます．

セミカラー・グラフィックスは，PC-8801の弟分であるPC-8001などが採用している方法で，隣

図3 フレーム・バッファ・ディスプレイ

図4 カラー表示の原理

ビットの状態			
緑	赤	青	表示される色
0	0	0	黒
0	0	1	青
0	1	0	赤
0	1	1	紫
1	0	0	緑
1	0	1	水色
1	1	0	黄色
1	1	1	白

表2 ビットの状態と表示される色の関係

接する何ドットかが同じ色となります．この方法の利点は，フルカラー・グラフィックスに比べて，メモリの容量が小さくて済むことですが，メモリICが安くなった昨今は使われなくなりました．

それに対し，フルカラー・グラフィックスは1点ごとに色が変更できるので，きめの細かい色指定ができたり，疑似的に中間色を作ったりすることができます．このようなカラー表示を行うには，光の三原色の各輝度情報を，それぞれフレーム・バッファに記憶している必要があります．そのため，1つのピクセルを表すには，最低3ビットは必要です．つまり，図4のように，青の色を表示するかしないかの情報を持つフレーム・バッファと，赤の色を表示するかしないかの情報を持つフレーム・バッファと，緑の色を表示するかしないかの情報を持つフレーム・バッファの3枚があれば，カラー表示が行えます．ディスプレイ・コントローラは，三原色1つ1つの輝度情報をそれぞれの色の電子銃に送り，それによって各ピクセルがカラーで表示されます．この時，表示される色は，表2のように，黒，青，赤，紫，緑，水色，黄色，白の8色です．

ここで，もっと柔軟な考え方をしてみましょう．nビットの2進数が表すことのできる状態は

図5　カラー・マップ・テーブルを用いたカラー表示

図6　マルチ・プレーン

グラフを切り換えて表示したり，重ねて表示したりする．

図7　マルチ・プレーンの使用例

2^n個ですから，その1つ1つの状態に，色を1つずつ対応させるのです．つまり，図5のように，色の成分表をフレーム・バッファとは別に持たせ，フレーム・バッファにはその表のうちどれを選択させるかの情報を持たせるのです．この色の成分表を，カラー・マップ・テーブル(color map table）といい，三原色の各色の輝度情報を記憶しています．図5では，フレーム・バッファの値が5なので，カラー・マップ・テーブルから5番の色の輝度情報を得て，それをアナログ量に変換して各色の電子銃に送っています．カラー・マップ・テーブルが書き込み可能ならば，さらに豊富な色を使えます．このように，カラー・マップ・テーブルを使いますと，フレーム・バッファの大きさに左右されずに，多彩な色の表現が可能となります．

1.1.3　マルチ・プレーン

ひとつのピクセル当りのビット数がたくさんあると，単に多くの色彩や濃淡で表示できるだけでなく，ちょっと違った表示も可能となります．それは，フレーム・バッファをいくつかのプレーン（plane，ページ（page）ともいう）に分割することによって，別々のプレーンに描かれた画像を重ねて表示できることです(図6)．例えば，ひとつのピクセル当りのビット数が6ビットだったら，3ビットごとの2枚のプレーンや，2ビットごとの3枚のプレーンに分割できます．このように複数のプレーンが得られると，図7のように，グラフなどが描かれたプレーンを切り換

2つのバッファを使って，動的表現を行う．

図8 マルチ・プレーンの使用例

図9 キャラクタのドット構成

えて表示したり，重ねて表示したりすることができます．さらに，図8のように，第1プレーンに書き込んでいる時，第2プレーンを表示することを繰り返せば，動的表現を滑らかに行うこともできます．

1.2 PC-8801の画面に関するハードウェア

1.1では，最も一般的なグラフィック装置であるグラフィック・ディスプレイについてお話ししましたが，ここでは，本書の対象機種であるPC-8801について，グラフィックスのために，どのようなハードウェアが搭載されているか見てみましょう．大まかなことは，第1巻の〝PC-8801 BASIC入門〟で取り上げていますので，ここでは，もっとハードウェアに寄った話をします．

1.2.1 テキストV-RAM

PC-8801の画面には，テキスト画面とグラフィック画面があることは，第1巻の〝PC-8801 BASIC入門〟などにも解説がありましたから，もう御存知のことと思います．テキスト画面の方は，文字通りリストなどのテキストを表示するための画面であり，本書の主題であるグラフィックスとは直接的に関係はありませんが，ドット・グラフィックスの画面にコメントを入れたり，簡単なキャラクタ図形を描いたりするのに使われます．ここでは，このテキストV-RAMが，どのような構成になっているかについてお話ししましょう．

☆キャラクタ表示

まず，1つのキャラクタ（例えば〝A〟など）は，どのような仕組みで表示されているか知っていますか？　PC-8801では，1つのキャラクタが，8×8ドットの大きさを1つの単位として構成されています(他のほとんどのパーソナル・コンピュータも同じ)．例えば図9のようになっています．但し，これで分かるように実際のキャラクタの表示に使っているのは，6×7ドットです．これは，キャラクタが並んだ時に，隣と接触しないようにするためです．

このようにキャラクタといえども，実際はドットで構成されていてドット・グラフィックスと似た所があります．違う所は，キャラクタの場合，あらかじめ用意されたパターン（文字など）

を決った位置（最大80×25の枠の中）にしか置けないということです．ドット・グラフィックスの場合では，640×200ドット（カラー・モード時）の任意の位置に任意のパターンを作ることができます．ちなみに，なぜ640×200なのかということも，ちょっと考えればすぐ分かります．8（ドット）×80（桁）=640，8（ドット）×25（行）=200だからです．

では，キャラクタを表示する場合，どのような手法が使われているのでしょうか？ まず，各キャラクタは，それぞれ固有のコードを持っています．これがいわゆるキャラクタ・コードなのですが，このコードをユーザーがBASICに与えることにより，表示させることができます．指定することのできるコードの数は，0から255までの256個で，この数は1Byte（8ビット）で表すことのできる最大の数（2^8）に起因しています．BASICは，このコードを受け取ると，そのコードに対応するキャラクタのパターン（例えば，図9のような8×8ドット（8Byte）パターン）をキャラクタ・ジェネレータ（通称キャラジェネ）という2KByteのROMの中から捜し，画面に表示するのです．

キャラクタは，このような手続きで表示される訳ですが，ユーザーはテキストV-RAMのエリアにキャラクタ・コードをセットするだけでよく，その処理のほとんどをハードウェアが行っています．このハードウェアの中枢が，CRTC（μPD3301D-2）というチップであり，その他，カーソルの制御などを担っています．

☆テキストV-RAMの構造

先程，説明した通り1つのキャラクタを表示するには，1バイトのデータを必要とする訳ですが，PC-8801のキャラクタ画面の最大値は，80×25=2000（キャラクタ）ですから，約2KByteのデータ領域を必要とすることになります．このキャラクタ表示用のデータ領域のことをテキストV-RAMといいます．メモリ・マップで示すと，図10の様になっており，このV-RAMの1Byteが画面の1キャラクタに反映されるのです．

図10 テキストV-RAMと画面の対応

☆キャラクタのカラー

PC-8801では，ドット・グラフィックスとは別に，キャラクタ単位でのカラー表示が可能です．先程テキストV-RAMは，2KByte必要だと説明しましたが，実際メモリ・マップを見てみると，3KByteのテキストV-RAMを持っています．この余分な1KByteがカラー表示のために使われている，アトリビュート（属性）エリアと呼ばれるものです．このアトリビュートに，各キャラクタのカラー情報を持っています．アトリビュート・エリアは，各行ごとに40Byteずつ，80Byteの表示用データ・エリアの後に続いています．アトリビュートは，2Byteで1つのキャラクタについての属性を持っており，結果的に1行20キャラクタまでの属性を保持することができます．PC-8801のテキスト画面において，1行中で20回までしかキャラクタ単位での色付けができないのは，このハードウェアの制約によるものです．

以上が，PC-8801のテキスト画面に関するハードウェアの概略ですが，この構成は，PC-8001の画面の構成と基本的に同じですから，詳しく知りたい方は，PC-8001関係の資料をお調べください．

1.2.2 グラフィックV-RAM

さて，本書の主題であるドット・グラフィックスです．1．2．1で説明したキャラクタは，1Byteで1つのパターンを表示できましたが，ドット・グラフィックスでは，1bitで1つの点を表示します．1．1で説明したようにそのbitが1の時は点が打たれ，0の時は点が消える仕組みなのです．PC-8801の分解能は，白黒3ページ・モードの時，640×200ドットですから，横方向に640bit（80Byte）必要であり，全画面では，80×200＝16000Byte（約16KByte）必要であることが分かります．この画面を3ページ持っている訳ですから，合計48KByteのグラフィックV-RAMを持っていることになります．このグラフィックV-RAMは，メモリ上では図11のように構成されています．

図11 グラフィックV-RAMのメモリマップ

この3つのグラフィックV-RAMが，色々組み合されてカラー・モードなどを生み出すことは，第1巻でもお話しした通りです．ここでは，グラフィックV-RAMが，CPUからどのように制御されているかをお話ししましょう．ここでの知識は，単にBASICを扱っているだけならば，さほど必要のないことですが，機械語を使ってグラフィカルなプログラムを作る時には，必要になってきます．具体的なアクセスの方法は，4章の「ツールの紹介」で解説しますから，ここではその概念を学んでください．

まず，図11のメモリ・マップを見ると分かるように，青，赤，緑の各V-RAMは，C000H～FFFFHの間でバンク構造をとっています．バンク構造とは層構造のことで，メインバンクを含めて4つの層になっており，それぞれのメモリバンクは共有のアドレスを持っています．例えば，C000Hというアドレスを持つメモリは，物理的には4個存在する訳です．この内，ドット・グラフィックスに使われるのはメインバンクを除いた3つのバンクです．そのうちのどれかをアクセスしたい場合は，アクセスする前にそのバンクを選択する必要があります．

では，なぜこのようなバンク構造になっているのでしょうか？　その答は簡単で，8bitのCPU（PC-8801は，Z80-Aという8bitCPU）が標準でアクセスできるメモリの数は，64KByteだからです．ですから，PC-8801のように180K以上のメモリを持つ機種の場合，そうせざるを得ないのです．このあたりは，8bitCPUの泣きどころとも言えるでしょう．

従って，実際に画面にカラーでドットをセットする場合は，このようにバンク切り替えにより，グラフィックV-RAMの各バンクを選択して行います．この場合，カラーでドットを表示するということは，どのカラー・バンクにドットをセットするかに相当します．

最初に述べたように，具体的な手法については，4章で解説しますから，ここでは，PC-8801のドット・グラフィックスが，上記したようなグラフィックV-RAMをバンク切り替えすることにより行われていることを覚えておいてください．

1.2.3　カラー・パレットとその他のカラー

PC-8801のグラフィックスの特徴として，カラー表示が多彩であることがあげられます．図12に示すように，キャラクタ，ドット，ボーダーと完全に個別に色付けが可能です．

キャラクタ
ドット
ボーダー

図12　画面のカラー

これらの仕組みは，第1巻及び前項で説明済みですから割愛させて頂きますが，機械語による操作の仕方は，4章で説明します．

1.2.4 入／出力機器

これまでは，PC-8801本体内のグラフィックス用のハードウェアについてお話ししてきましたが，ここでは，CRTなどの入/出力機器について見てみましょう．

☆CRTディスプレイ

CRTディスプレイは，1．1でもお話しした通り，グラフィックスを行う上で最も一般的な出力機器です．PC-8801に接続可能なCRTとしては，NECから，高解像度CRT(640×400)のPC-8853(カラー）とPC-8851（モノクロ)，640×200のCRTとして，PC-8049K（カラー）とPC-8059K（モノクロ）が発売されていますので，ここでは，これらの機種を対象とします．

CRTの場合，考慮しなければならないことは，解像度と発色の問題でしょう．まず，発色に関して問題になるのは，当然カラーの方です．PC-8801から出てくる信号は，RGB信号ですから，黒，青，紫，赤，緑，水色，黄色，白の8色が表示できます．逆にいえば，テレビのようにその他の中間色を発色することはできないのです．また，1．2．2で説明したように，各3原色に対して，1つのプレーンしか持っていませんから，淡い青とか濃い青のように明度を変えることもできません．この問題は，NEC以外のCRTについてもいえることです．

次に解像度ですが，これはグラフィックスを行う上で一番の問題になっています．解像度が大きいということは，細かい図形が描ける訳ですから，なるべく大きい方がよいのです．先程紹介した中でいえば，PC-8853，PC-8851の方が有利なのです（但し，値段は高くなりますが)．

もう1つ解像度において非常に重要な問題があります．それは，1ドットの縦・横比です．これは，グラフィックスを行う上で，かなりのネックになります．というのは，CRTの大きさと分解能の関係によって，縦・横比が1：1にならないものがあるからです．図13はPC-8853とPC-8049Kのドットです（但し，ここであげた数値はCRTの製造誤差などの理由により，完全なものとは言えませんが，実用上問題ないと思われます．また，このように整数比になっていることは，いろいろな面で有利です)．

従って，縦・横比の異なるCRT上で正方形などを描こうとする場合，必ず補正を行わなければならないのです．補正の方法は後述しますが，ここでは，画面に表示される図形は補正しないと歪んでいることを覚えておいてください．

その他のCRTをお使いの方は，自分で画面のサイズを測って，縦・横比を算出してください．

☆プリンタ

一般にプリンタは，グラフィックスの出力機器としては使われませんが，最近のプリンタはドット・イメージがプリントできるため，画面のハード・コピーを取るのによく使われています．CRT

図13　1ドットの縦・横比

のコピーなので，特に問題はありませんが，プリンタにおいても縦・横比が違いますから，その分，考慮しておかなければなりません．ここでは，NEC純正のPC-8821/8822とPC-8023Cについて縦・横比を示しておきます（純正以外のプリンタの場合，そのままではハード・コピーは取れません）．

PC-8821／8822	COPY2	→	x：y＝1：2.7
	COPY4	→	x：y＝1：1.3
PC-8023C	COPY2	→	x：y＝1：2.2
	COPY4	→	x：y＝1：1.1

☆入力機器

グラフィックスでは，図形データを入力する入力機器もまた重要です．しかし，残念なことにNEC純正のシステムには，それに類するものはありません．最もてっとり早い方法は，グラフ用紙に図形を描いて座標データを目で読み取り，キーボードより入力する方法ですが，この方法では複雑な図形になればなるほど労力と時間が必要になってきます．そこで本書では，複雑な図形の入力機器にあたるものを，キーボードとCRTとソフトウェアによって作り出し，図形データを取り込んでいます（4章で解説します）．

2章　基本図形を描く

基本図形，例えば，点，線，四角形，円を任意の位置に，任意の大きさで描くことができることは，コンピュータ・グラフィックスの基本です．この基本をマスターすることが，この章の課題です．点，線，四角形，…，これらは，1つの点，または複数の点を結ぶことによって描くことができます．そのため，この章では，まず座標についてお話しします．次に，点，線，…と順に描く手法をお話ししますが，単にN_{88}-BASICの命令の紹介だけでなく，いくつか一般に使われているアルゴリズムについてもお話しします．これらのいくつかについては，第1巻でも解説してありますが，ここではさらに詳しく解説します．

ところで読者の中には，「N_{88}-BASICには，直線や円を描く命令が最初からあるのだから，わざわざ別の方法でやることはないのでは」とお思いの方もいらっしゃるでしょう．では，次のような画面を作りたいとします．どうしますか？

図1　円周上の三角形

この図は，三角形をある点を中心とした円周上に何個も描いたものです．この図形をコンピュータで実現するには，どうしても計算によって円周上の点の座標を求める必要があるのです．このような用途の場合，CIRCLE命令など何の役にも立ちません．この章で解説するいくつかのテク

ニックをマスターすることによって，この例のような色々な応用図形を描けるようになるはずです．

それともう1つ，グラフィックスを本格的にやっていこうとすると，BASICではどうしても速度への不満が生じてくるでしょう．そこで，機械語による高速なプログラムが要求されます．この章で紹介する数々のアルゴリズムは，その場合にも，きっとお役に立つことでしょう．

2.1 座標系と点

点を打つには，その点を打つ位置を的確に表し，指定する必要があります．これには，〝座標〞を使います．ここでは，この座標についてお話しします．

2.1.1 座標って何

座標とは，空間あるいは平面上の位置を表す数の組のことです．身近な例を2つ上げてみましょう．碁盤では，その上にある石の位置を，縦と横の線を組み合わせて，5の四などといいます．また，地図帳を広げると，縦横に経線・緯線が走っていて，例えば，東京の位置を北緯35度40分東経139度46分と表します．

解析幾何学では，図4のように，原点Oで直角に交わる2本の座標軸（横にX軸，縦にY軸）を使って，点Pを（x，y）と表します．この時，xは点PからX軸におろした垂線とX軸の交点の目盛りで，〝x座標〞といいます．そして，yは点PからY軸におろした垂線とY軸の交点の目盛りで，〝y座標〞といいます．このような座標系を直交座標系，あるいはデカルト座標系といいます．直交座標系では，xの値は左から右へ，yの値は下から上へ増加します．

このほかの座標系としては，3次元の直交座標系，極座標系などがあります．

図2 碁盤(座標の例)

図3 地図(座標の例)

図 4　直交座標

2.1.2 グラフィック座標系

グラフィック画面上で位置を指定するには，今お話しした直交座標の考え方を使います．点は x 座標と y 座標によって，（x，y）と表します．x の値は直交座標系と同じく左から右へ，y の値は直交座標と逆に上から下へ増加します．N_{88}-BASIC のグラフィックスの分解能は，640×200 と640×400の 2 つですから，各モードの座標系はそれぞれ図 5 の左，右となります．この 2 つの座標系を総称して，グラフィック座標系といいます．これは，テキスト画面のキャラクタ座標系に対して言う時によく使われます．グラフィック座標系の範囲外を指定した場合は，表示されなかったり，エラーが起きたりします．また，x の値や y の値の精度には限界がありますので，2 つの別々の座標を与えたからといって，必ずしも画面上で違う位置を指定しているとは限りません．これは，画面上の点の数が有限であることに基づいています．図 5 の場合は，点の間隔が 1 ですので，x の値や y の値の小数点以下は意味を持ちません．但し，小数第 1 位は四捨五入されます．

注意：図 5 は後述するワールド座標系，スクリーン座標系に対して，オリジナル・スクリーン座標系と呼ばれます．点を打ったり，線を引いたりするとき座標系は，ワールド座標系ですが，BASIC の起動時は，ワールド座標系とオリジナル・スクリーン座標系は等しいと考えられますので，この章では図 5 の座標系を使ってお話ししていきます．

図 5　N_{88}-BASICのグラフィック座標系

2.1.3 絶対座標と相対座標

今までお話ししてきた座標の指定は，〝絶対座標〞によるものでした．絶対座標では，x座標，y座標が決まることによって一意的に座標が決まります．例えば，図5の左のような座標系があったとすると，左上の頂点は常に（0，0）で，右下の頂点は常に（639,199）となり（320,100）といったら，ちょうどまん中になります．〝絶対座標による指定〞とはこの絶対座標を使って位置を表す方法をいいます．N_{88}-BASIC での座標指定は，普通どの座標系（キャラクタ座標系，グラフィック座標系）に対してもこの指定により行います．但し，点を打つ，線を引くなどの座標指定には，〝相対座標〞を使うことができます．

タバコ屋までの道を聞かれた時，「××　△丁目○ですよ」などと答える人はいないでしょう．指をさして，「この道を60m程行った所の交差点を右に曲がって，20m程行くと右側にありますよ」などというでしょう．このように，道を教える時は現在の位置を基準にして，指をさしたり，右とか左とかいったりして方角を表し，60m程などといって歩く距離を表すでしょう．これが相対座標の考え方です．N_{88}-BASIC では，現在の位置を，Last referenced Point（以降〝LP〞と略す）として持っています．このLPは最後に参照された点という意味で，グラフィック命令で座標を与えた場合，最後に指定された座標を値として持ちます．方角と距離は，x，yの符号と数値によって表されます．相対座標で(10,−10)と表された点P'は，基準点Pからxの正方向に10,yの負方向に10のところにあります（図7）．

図6　相対座標の考え方

図7　相対座標による指定

基準点が絶対座標で(320,100)に設定されていると，図8のような相対座標系を作ることができます．この時，左上の頂点の座標は（−320,−100）となり，右下の頂点は（+319,+99)となります．基準点から相対移動量(+160,−40)だけ移動した点は，絶対座標では（480,60）となります．ここに基準点(LP)を移動させると，座標軸は点線の位置となり，今までと違った相対座標系を作ります．

図 8　相対座標

この方法の優れている点は，相対移動量を表す（x，y）は常に同じ値であっても，基準点の位置を動かすことによって違った位置を簡単に表現することができることです．

N_{88}-BASIC のグラフィック命令で相対座標を使って座標指定を行う場合は，

```
STEP (X, Y)
```

のように，座標を表す（X，Y）の前に〝STEP〞を付けて（X，Y）が相対量であることを表します．

N_{88}-BASIC のグラフィック命令で相対座標の指定が許されているのは，

PSET
PRESET
LINE
PAINT
CIRCLE
POINT（ステートメント及び関数）
GET＠(但し終点座標のみ)

の 7 つです．

2.1.4　点を打つ命令PSET文

点を表示するステートメントが PSET（ピーセット）です．PSET 文の書式は，

```
PSET (X, Y), C
```

です．X，Yは点を表示する位置で，グラフィック座標（ワールド座標）で指定します．（X，Y）の前に STEP を付けると相対座標による指定となります．

Cは色の指定で，パレット番号により指定します．色の指定が無い時は，現在のフォア・グラウンド・カラーで表示します．

LP は，指定された座標（X，Y）に設定されます．

次に例を載せておきますので，実行させて確かめてみてください．

```
pset(200,10)                 ← 白で点がプロットされる
Ok
pset step(-10,10),4 ← そのすぐ左下に緑で点がプロットされる
Ok
```

（色は BASIC 起動時の初期設定に合わせてありますので，表示される色が違う場合があるかもしれません）．

2.1.5 点を消す命令PRESET文

PSET 文と逆の動作，すなわち表示している点を消去するのがPRESET（ピーリセット）です．

PRESET (X, Y), C

座標の指定は PSET 文と同じです．普通は色の指定をするCを省略します．この時，(X, Y) で指定された位置のドットは，現在のバック・グラウンド・カラーで塗り変わります．

Cを指定した時は，PSET 文と全く同じに機能し，そのパレット番号のカラーで点をセットします．

LPは指定された座標（X, Y）に設定されます．

次の例を実行してみてください．

```
pset(200,10)           ← 点が白で表示される
Ok
preset(200,10),4 ← 緑に変わる
Ok
preset step(0,0) ← 消える
Ok
```

（色はBASIC起動時の初期設定に合わせてありますので，表示される色が違う場合があるかもしれません）．

2.1.6 夜空—乱数を使って点を打つ—

点を1つだけ打ったり消したりしているのではおもしろくないので，1つプログラムを作りました．乱数を使って点の位置と色を決めて，点を打つプログラムです．点の数はNで設定できます．100個ぐらいが適当でしょう．リスト1を実行しますと，夜空に星が点滅しているように見えます．

リスト1

```
100 '--  STAR NIGHT  --
110  INPUT "ﾎｼ ﾉ ｶｽﾞ";N
120  SCREEN 0,1:CLS 3:SCREEN 0,0
130  DIM STAR(N,1)
140 *LOOP '--< star draw >--
150  FOR I=1 TO N
160    PRESET(STAR(I,0),STAR(I,1))
170    STAR(I,0)=RND*639.5
180    STAR(I,1)=RND*199.5
190    PSET(STAR(I,0),STAR(I,1)),INT(RND*7+1)
200  NEXT
210 GOTO *LOOP
```

2.1.7 LPを設定するPOINT文

POINT（ポイント）文は，点の表示や消去に関係なくLP を設定する命令です．

POINT (X, Y)

座標の指定は PSET 文と同じです．LP は指定された座標に設定されます．この POINT 文は，相対座標で図形を描く時，その基準点を設定するのに使われますので，覚えておいてください．

```
pset(160,50)          ←― 左上に点が白で表示される
Ok
point step(160,50)←― まん中にLPを設定する
Ok
pset step(160,50)  ←― 右下に点が白で表示される
Ok
```

2.2 直線で図形を描く

直線は，コンピュータ・グラフィックスで最もよく使われます．棒グラフ，折れ線グラフ，ブロック・ダイヤグラム，フロー・チャート，文字，論理素子などさまざまな図に必ず直線が見られます．その上，曲線は，それがつながって見えるように，とても短い直線によって描きます．このように，任意の2点間を直線で結ぶことは，あらゆる絵をコンピュータで描く上での基本となっています．そこで，ここではまず直線を引く命令を覚えて，次にその命令を使って基本的な図形――三角形，四角形など――を描いてみましょう．

2.2.1 直線を引くには

直線を引く命令は，市販のパーソナル・コンピュータには必ずといっていい程付いている機能です．もちろん，N_{88}-BASIC にも例外なく付いています．しかし，その直線を引く命令のアルゴリズムを知りたい方もいるかと思いますので，ここではそのアルゴリズムをいくつか紹介します．もちろん，いちはやく直線を引く命令を知りたい方は，ここを読み飛ばして2．2．2から読んでも構いません．

さて，直線を描くにはどうしたらよいでしょうか．まず思いつくのは，直線の方程式でしょう．線分の始点と終点が与えられれば，その線分の方程式が求められます．図9のように，始点を(x_0, y_0)，終点を(x_1, y_1)とすると，直線の方程式は次のように表されます．

$$\frac{y-y_0}{y_1-y_0}=\frac{x-x_0}{x_1-x_0} \qquad \cdots\cdots(1)$$

この(1)式を $y=$ の形に変形すると，

$$y = \frac{y_1 - y_0}{x_1 - x_0}(x - x_0) + y_0 \qquad \cdots\cdots\cdots\cdots (2)$$

となります．ここで，xの値をx_0からx_1まで変化させてyの値を求めて，(x, y)の座標に点をプロットすれば，任意の線分が描けます．

図 9　直線の考え方

(2)式を使ってプログラムを組むと，リスト2の様になります．

リスト2

```
100 '--  DRAW LINE (1)  --
110  SCREEN 0,0:CLS 3
120  INPUT "X0,Y0 ";X0,Y0
130  INPUT "X1,Y1 ";X1,Y1
140  A=(Y1-Y0)/(X1-X0)
150  FOR X=X0 TO X1 STEP SGN(X1-X0)
160    Y=A*(X-X0)+Y0
170    PSET(X,Y)
180  NEXT
190 END
```

ところが，このプログラムには問題があります．(2)式で$x_1 - x_0$が0のとき，つまり引こうとする線がy軸に平行なときは，分母が0となってしまうからです．また，次のように立った線では，プロットした点が離れてしまって，点線となってしまいます．

そこで，リスト2に工夫を加えて，(2)式の分母が0になったり線が点線になったりするのを防ぐことにします．分母が0になったり線が点線になったりするときは，分母より分子が大きい，つまり，

$$|x_1-x_0| \quad < \quad |y_1-y_0| \qquad \cdots\cdots(3)$$

の条件が成り立つときです．このときは，(2)式の代わりに，

$$x=\frac{x_1-x_0}{y_1-y_0}(y-y_0)+x_0 \qquad \cdots\cdots(4)$$

を使って，yの値を変化させてxの値を得れば，先程の問題は解決されます．こうした修正を加えたプログラムがリスト3です．

リスト3

```
100 '--  DRAW LINE (2)  --
110  SCREEN 0,0:CLS 3
120  INPUT "X0,Y0 ";X0,Y0
130  INPUT "X1,Y1 ";X1,Y1
140  IF ABS(X1-X0)<ABS(Y1-Y0) THEN *BIGY
150  A=(Y1-Y0)/(X1-X0)
160  FOR X=X0 TO X1 STEP SGN(X1-X0)
170    Y=A*(X-X0)+Y0
180    PSET(X,Y)
190  NEXT
200 END
210 *BIGY
220  A=(X1-X0)/(Y1-Y0)
230  FOR Y=Y0 TO Y1 STEP SGN(Y1-Y0)
240    X=A*(Y-Y0)+X0
250    PSET (X,Y)
260  NEXT
270 END
```

さて，リスト3を実行すると分かるように，線を描くのが実に遅いですね．写真のように1本の線を引くだけならあまり気になりませんが，何十本もの線を引くとしたら膨大な時間を必要とします．そこで機械語等で高速に処理させることが考えられます．ところが，機械語等を使ってプログラムを書いたとしても，実数を扱かったり，掛け算や割り算を計算したりするのでは，大幅な改善はみられません．そこで，整数だけを扱う方法を考えることにします．

今までと同じように線分の始点，終点を $(x_0,\ y_0)$，$(x_1,\ y_1)$ とします．そして，

$$\Delta x = x_1-x_0$$
$$\Delta y = y_1-y_0 \qquad \cdots\cdots(5)$$

とします．

リスト3をじっくり見ると，Δx と Δy の大きい方を1ずつ変化させて，他方の座標値は真の値に近い整数値を選んでいることが分かります．これは，グラフィック画面に線を表示するためには，点に分解しなければならないうえ，その点の打てる位置が限定されていることに起因します．ここで話を簡単にするため，$0<\Delta y<\Delta x$ と仮定します．この場合，初期値を，

$$x(0)=x_0 \qquad \cdots\cdots(6)$$
$$y(0)=y_0$$

とすると，

$$x(i)=x(i-1)+1 \qquad \cdots\cdots(7)$$
$$y(i)=y(i-1)+\Delta y/\Delta x$$

となります．

ちょっと待ってください．割り算がでてきましたね．割り算の答は実数になりますね．そうすると，先程言った「整数だけの方法」に反してしまいます．そこでいくつか工夫を加えることにします．

y (i)はその整数の部分のみ有効なので，y (i)を整数部と小数部に分けて，前者をy (i)で，後者をe (i)とすると，

$$e(i)=e(i-1)+\Delta y/\Delta x \qquad \cdots\cdots(8)$$
但し，$e(0)=0$

となり，

$e(i)<0.5$ なら $y(i)=y(i-1)$ $\qquad \cdots\cdots(9)$
$e(i)\geqq 0.5$ なら $y(i)=y(i-1)+1$，$e(i)=e(i)-1$

で，y (i)が決定されます．ここで1で比較しないで，0.5で比較した理由は，図10を見れば一目瞭然でしょう．

ここで，e (i)をE (i)＝e (i)－0.5とすれば，(9)式は符号の制定だけで済むので，アルゴリズムが簡単になります．つまり，(8)，(9)式は次のようになります．

(a) eを0.5と比較した場合

(b) eを1と比較した場合
(a)のほうがよい

図10

$$E(i)=E(i-1)+\Delta y/\Delta x$$
但し，$E(0)=-0.5$ ……………………………(10)

$E(i)<0$ なら $y(i)=y(i-1)$
$E(i)\geqq 0$ なら $y(i)=y(i-1)+1$，$E(i)=E(i)-1$ ……………………………(11)

このアルゴリズムをフローチャートで表せば，図11のようになります．

このアルゴリズムでは，X_0，Y_0，X_1，Y_1 が整数なので，X，Y，DX，DY は整数となります．しかし，Eは実数であり，また割り算が必要です．ところが，Eはその符号のみが必要とされま

BEGIN
Y←Y0
X←X0
E←−0.5
点(X, Y)を
プロットする
X=X1 ?
YES
END
NO
X←X+1
E←E+DY/DX
E≧0 ?
NO
YES
Y←Y+1
E←E−1

始点座標(X0, Y0)
終点座標(X1, Y1)
DX←X1−X0
DY←Y1−Y0

図11 アルゴリズム(1)

BEGIN
Y←Y0
X←X0
R←−DX
点(X, Y)を
プロットする
X=X1 ?
YES
END
NO
X←X+1
R←R+2・DY
R≧0 ?
NO
YES
Y←Y+1
R←R−2・DX

図12 アルゴリズム(2)

すので，Eの代わりにEを2・DXしたRを使えば，すべての変数が整数で扱える上に割り算がなくなります．まとめると次のようになります．

$$\begin{cases} x(0)=x_0 \\ y(0)=y_0 \\ R(0)=-\Delta x \end{cases} \quad \cdots\cdots(12)$$

$$\begin{cases} x(i)=x\ (i-1)+1 \\ R(i)=R\ (i-1)+2\cdot\Delta y \\ R(i)<0 \ \text{の時}\ \ y(i)=y\ (i-1) \\ R(i)\geqq 0 \ \text{の時}\ \ y(i)=y\ (i-1)+1,\ R(i)=R(i)-2\cdot\Delta x \end{cases} \quad \cdots\cdots(13)$$

○ xの値は前のxの値に1を加えたものである．

○前のRに2倍のΔyを加えたものが，新しいRとなる．

○ Rの値が負の時はyの値は変わらないが，Rの値が正または0の時は前のyの値に1を加えたものが新しいyの値となる．この時，Rからは2倍のΔxを引いておく．

このアルゴリズムをフローチャートで表すと，図12になります．

さて，ここまでは $0<\Delta y<\Delta x$ を仮定してきましたが，$\Delta x<\Delta y$ の時はどうしたらよいでしょうか．もう分かりましたね．また，$x_0>x_1$ のときや $y_0>y_1$ のときは，1を加えるところを1を減らしてやればよいのです．このアルゴリズムを元にBASICで作ったプログラムをリスト4に示します．実際にこのアルゴリズムを実行する時は，機械語やハードウェアで行います．

リスト4

```
100 '--  Bresenham Method  --
110  SCREEN 0,0:CLS 3
120  INPUT "X0,Y0 ";X0,Y0
130  INPUT "X1,Y1 ";X1,Y1
140  DX=X1-X0:DY=Y1-Y0
150  X=X0:Y=Y0
160  A=2*ABS(DX):B=2*ABS(DY)
170  IF A<B THEN *BIGY
180  R=-ABS(DX)
190  FOR I=0 TO ABS(DX)
200    PSET(X,Y)
210    X=X+SGN(DX)
220    R=R+B
230    IF R>=0 THEN Y=Y+SGN(DY):R=R-A
240  NEXT
250 END
260 *BIGY
270  R=-ABS(DY)
280  FOR I=0 TO ABS(DY)
290    PSET(X,Y)
300    Y=Y+SGN(DY)
310    R=R+A
320    IF R>=0 THEN X=X+SGN(DX):R=R-B
330  NEXT
340 END
```

このような方法をDDAといいます．DDAとは，デジタル微分解析機（Digital Differential Analyser）の略で，コンピュータ・グラフィックでの図形処理に使われる方法です．原理や詳しい内容については専門書に譲ります．なお，ここでとりあげたDDAの方法は，Bresenhamの

方法と呼ばれるものです．このほか，単純形 DDA や対称形 DDA が知られています．

2.2.2 直線を引く命令LINE文

N_{88}-BASIC で直線を引く命令は，LINE（ライン）文です．LINE 文は，

LINE (x_0, y_0)－(x_1, y_1)，色，オプション，ラインスタイル

という形式で使います．(x_0, y_0)は始点座標，(x_1, y_1)は終点座標で，共にグラフィック座標（ワールド座標）で扱います．色はパレット番号によって指定し，引く線の色を決めます．もしこれが省略されたら，COLOR 文によって設定されている，グラフィック画面のフォア・グラウンド・カラーが採用されます．命令実行後，LP は終点座標に設定されます．

```
line(50,10)-(300,100)←白で左上から右下へ線が引かれる
Ok                     (但し，BASICの処理上の理由で，始点から終点に向かって
line(320,180)-(320,50),4                              線が引かれるとは限らない)
Ok                     └緑で下から上へ線が引かれる
```

始点座標 (x_0, y_0) が省略されると，LP が始点座標として採用されます．

```
line-(530,50),1 ← 青で，まん中から右へ線が引かれる
Ok
```

始点と終点の指定には，PSET や PRESET と同様に相対座標も使えます．

```
line(100,50)-step(-50,25) ← 画面の左上に，左下へ直線が引かれる
Ok
line-step(100,0)          ← 続けて，右へ線が引かれる
Ok
```

今までのようにオプションを指定しないと，始点と終点を結ぶ直線を引きますが，オプションに B を指定すると，始点座標 (x_0, y_0) と終点座標 (x_1, y_1) を対角とする箱を書きます．

```
line(0,0)-step(160,50),4,b ← 画面の左上に緑で箱を書く
Ok
line-(320,100),5,b         ← そのすぐ右下に水色で箱を書く
Ok
```

また，オプションに BF を指定すると，箱の中まで塗りつぶします．

```
line step(-80,-25)-(400,125),6,bf
Ok                     └画面のまん中に黄色で箱を書き，その中を塗りつぶす
line-step(160,-50),3,bf ← そのすぐ右に紫で箱を書き，その中を塗りつぶす
Ok
```

書いた直線や箱を消すにはどうするか分かりますか．色にグラフィックの〝バック・グラウン

ド・カラー″のパレット番号——一般に0（黒）を指定すればよいのです．

```
line(0,0)-(160,50),0,bf ←左上の箱が消える
Ok
line step(80,25)-step(160,50),0,bf
Ok                          ↑まん中の箱が消える
```

ここで，LINE命令のサンプルとして，ちょっときれいなプログラムを紹介しておきましょう．

リスト5

```
100 '--  EXAMPLE  --
110  SCREEN 0,3:CLS 3:SCREEN ,0
120  WIDTH 80,25:CONSOLE 0,25,0,1
130  N=40 : XMAX=639 : YMAX=199
140  XS=XMAX/N : YS=YMAX/N
150 '--  line draw  --.
160  FOR X=0 TO XMAX STEP XS
170    LINE(XMAX-X,0)-(X,Y),7
180    Y=Y+YS
190  NEXT
200  Y=YMAX
210  FOR X=XMAX TO 0 STEP -XS
220    LINE(XMAX-X,YMAX)-(X,Y),4
230    Y=Y-YS
240  NEXT
250  X=0
260  FOR Y=0 TO YMAX STEP YS
270    LINE(0,YMAX-Y)-(X,Y),2
280    X=X+XS
290  NEXT
300  X=XMAX
310  FOR Y=YMAX TO 0 STEP -YS
320    LINE(XMAX,YMAX-Y)-(X,Y),1
330    X=X-XS
340  NEXT
350 END
```

2.2.3　三角形，四角形，五角形，…

グラフィックの特徴は，数式で書かれた何だか分からないものを，視覚に訴えることのできる具体的な形で表せることです．そこで，2.2.2では直線を引く命令を覚えましたので，これを使って，簡単な図形を描いてみましょう．直線で描ける幾何図形は次のように分類できます．

三角形……正三角形，直角三角形，二等辺三角形，不規則三角形など．

四角形……正方形，長方形，菱形，台形，不規則四角形など．

多角形……五角形，六角形，その他多角形，正多角形，星形など．

これらの図形を実際に描くにはどうしたらよいでしょうか．「コンピュータ君，画面のまん中に正三角形を書いてくれ」などと言っても，コンピュータは何もやってくれません．図形をコンピュータに描かせるには，次の要素を具体的に考えて，コンピュータに理解できる言葉——BASICで書かなければなりません．

どんな形を………三角形，四角形，それらを組み合わせた形．

どんな色で………1色の場合，多色の場合はどんな色の組み合わせで．

どんな大きさで…大小，大きさを変化させる．

どの位置に………右上，まん中，左下．

どの向きで………上向き，下向き，右向き．

(1) 三角形

三角形は直線3本によってできていますので，LINE 文3回で三角形が書けるはずです．先程述べた各要素から三角形の3つの頂点の座標を得て，それらの頂点をお互いに結べばよいのです．リスト6が，三角形を書くプログラムです．170行のDATA文で，正三角形の3つの頂点の座標を持っています．そのため，このプログラムを実行すると，正三角形が左上の方に描かれます．もちろん，170行のDATA文を変えれば，いろいろな形の三角形が描けます．

リスト6

```
100 ' --  Triangle  --
110  SCREEN 0,0:CLS 3
120  READ X0,Y0,X1,Y1,X2,Y2
130  LINE(X0,Y0)-(X1,Y1)
140  LINE-(X2,Y2)
150  LINE-(X0,Y0)
160 END
170 DATA 80,10,0,80,160,80
```

さて，絵グラフなどでよくあることは，同じ形の図形を位置や大きさを変えて表示することです．8章でお話しする2次元変換を使ってもできますが，位置を変えて表示する時は，相対座標を使った方が簡単に行えます．リスト7が相対座標を使ったプログラムです．300行から360行が三角形を描くサブルーチンです．このサブルーチンを呼ぶ前には，三角形を描く位置を決める基準点と，三角形の大きさを決めるNを与えなければなりません．基準点は図13のように設定してあって，POINT 文で LP として与えるようになっています．倍率Nは，変数Nに代入して与えます．

リスト7

```
100 '--  EXAMPLE (1)  --
110  SCREEN 0,0:CLS 3
120 *LOOP
130  N=RND*320
140  C=RND*7+.5
150  X=RND*639.5
160  Y=RND*199.5
170  RESTORE
180  POINT(X,Y)
190  GOSUB *TRI
200 GOTO *LOOP
300 '--<  Triangle  >--
310 *TRI
320  READ X0,Y0,X1,Y1,X2,Y2
330  LINE STEP(N*X0,N*Y0)-STEP(N*X1,N*Y1),C
340  LINE-STEP(N*X2,N*Y2),C
350  LINE-STEP(-N*(X1+X2),-N*(Y1+Y2)),C
360 RETURN
370 DATA .5,0,-.5,.433,1,0 ←— 三角形用データ (640×200用)
```

図13 三角形と基準点

310行から370行のプログラムを説明しておきましょう．基準点からみた三角形の各々の頂点を図13のように (x_0, y_0)，(x_1, y_1)，(x_2, y_2) として，線はNo. 1→No. 2→No. 3→No. 1と描くことにします．2つの点の間のx座標とy座標の差を求めれば，ある点からみた次の点の相対座標が決まりますね．

基準点→No.1　(図14(a))

$$\Delta x_0 = x_0 - 0 = x_0$$

$$\Delta y_0 = y_0 - 0 = y_0$$

No. 1→No. 2　(図14(b))

$$\Delta x_1 = x_1 - x_0$$

$$\Delta y_1 = y_1 - y_0$$

No. 2→No. 3　(図14(c))

$$\Delta x_2 = x_2 - x_1$$

$$\Delta y_2 = y_2 - y_1$$

No. 3→No. 1　(図14(d))

$$\begin{aligned}\Delta x_3 = x_0 - x_2 &= x_0 - x_2 + x_1 - x_1 \\ &= -(x_1 - x_0) - (x_2 - x_1) \\ &= -\Delta x_1 - \Delta x_2\end{aligned}$$

$$\begin{aligned}\Delta y_3 = y_0 - y_2 &= y_0 - y_2 + y_1 - y_1 \\ &= -(y_1 - y_0) - (y_2 - y_1) \\ &= -\Delta y_1 - \Delta y_2\end{aligned}$$

Δx_3をΔx_1，Δx_2で，Δy_3をΔy_1，Δy_2で表したのは，DATA文でのデータの個数を減らすためです．

そして，これらのデータを使って三角形を描くには，

図14　相対座標による三角形

LINE−STEP $(\Delta x_0,\ \Delta y_0)$−STEP $(\Delta x_1,\ \Delta y_1)$

LINE−STEP$(\Delta x_2,\ \Delta y_2)$

LINE−STEP$(-\Delta x_1 - \Delta x_2, -\Delta y_1 - \Delta y_2)$

とすれば良いのです．

例として，図15のような正三角形を取り上げてみましょう．図の値から計算すると，

図15　正三角形のデータ

$\Delta x_0 = 0.5$, $\Delta y_0 = 0$

$\Delta x_1 = -0.5$, $\Delta y_1 = 0.866$

$\Delta x_2 = 1$, $\Delta y_2 = 0$

となります．この値は640×400の高分解能白黒モード用で，640×200の時は1.2.4で説明した通り補正する意味で，yの値を2で割った値を使ってください．これは，640×200の時のドットの縦・横比が約2：1であることに起因しています．

直角三角形を描きたい時は，

```
DATA .5,0,-.5,.25,1,0
```

を，二等辺三角形を描きたい時は，

```
DATA .5,0,-.5,.5,1,0
```

を，使ってください（640×200用）．

(2) 四角形

四角形はLINE文のBオプションでも書けますが，(1)三角形と同様に考えますと，リスト8の様になります．480行のDATA文を変えることによって，いろいろな四角形が描けます．

リスト8

```
100 '--  EXAMPLE (2)  --
110  SCREEN 0,0:CLS 3
120 *LOOP
130  N=RND*320
140  C=RND*7+.5
150  X=RND*639.5
160  Y=RND*199.5
170  RESTORE
180  POINT(X,Y)
190  GOSUB *SQ
200 GOTO *LOOP
400 '--<  Square  >--
410 *SQ
420  READ X0,Y0,X1,Y1,X2,Y2,X3,Y3
430  LINE STEP(N*X0,N*Y0)-STEP(N*X1,N*Y1),C
440  LINE-STEP(N*X2,N*Y2),C
450  LINE-STEP(N*X3,N*Y3),C
460  LINE-STEP(-N*(X1+X2+X3),-N*(Y1+Y2+Y3)),C
470 RETURN
480 DATA .3,0,.7,0,-.3,.3,-.7,0 ←─ 平行四辺形用データ（640×200用）
```

菱形用データ（640×200用）

```
DATA .5,0,.5,.15,-.5,.15,-.5,-.15
```

台形用データ（640×200用）

```
DATA .3,0,.4,0,.3,.3,-1,0
```

基準点 x_0, y_0 x_1, y_1
$-x_1-x_2-x_3$
$-y_1-y_2-y_3$
x_2, y_2
x_3, y_3
N

図16 相対座標による四角形

(3) **多角形**

多角形も三角形や四角形と同様に描けます．しかし，頂点の数が増えると，最後の線を結ぶ時の計算式が長くなってしまいます．そこで，POINT 関数を使って，プログラムに工夫を加えましょう．

POINT 関数は2種類ありますが，ここではLPの位置を与える方を使います．この POINT 関数は，

POINT（機能）

のように用います．機能は0から3までの整数値をとり，それぞれの場合にPOINT関数の返す値の意味は次の通りです．

POINT(0)：LP のX座標を返します（ワールド座標系の値で）．
POINT(1)：LP のY座標を返します（ワールド座標系の値で）．
POINT(2)：スクリーン座標系（後述）の値で，LPのX座標を返します．
POINT(3)：スクリーン座標系の値で，LPのY座標を返します．

リスト9は POINT 関数を使った正五角形を描くサブルーチンです．リスト8で，400行から480行を削除して，リスト9を加えてください．この時，

```
190 GOSUB *PENT
```

と変更してください．

リスト9では，560行で POINT 関数で得た LP を覚えていて，600行で第1点と第5点を線で結ぶのに使っています．これによって，最後の線を結ぶ時の計算式を省いています．

リスト9

```
500 '--  Pentagon  --
510 *PENT
520  FOR I=0 TO 4
530    READ X(I),Y(I)                    } 各点の相対座標をデータから読み込む
540  NEXT
550  POINT STEP(N*X(0),N*Y(0))      ←― 第1点にLPを移す
560  X(5)=POINT(0):Y(5)=POINT(1) ←― LPを読む
570  FOR I=1 TO 4
580    LINE -STEP(N*X(I),N*Y(I)),C } 第1点から第5点まで線で結ぶ
590  NEXT
600  LINE -(X(5),Y(5)),C      ←― 第5点と第1点を線で結ぶ
610 RETURN
620 DATA .476,0,-.476,.173,.155,.280 } 正五角形用データ（640×200用）
630 DATA .697,0,.155,-.280
```

620行と630行の DATA 文を，

```
620 DATA .476,0,-.321,.452,.797,-.280
630 DATA -.951,0,.797,.280
```

と変えると，星形が描けます．

せっかく多角形の話ですから，ここで，DATA 文を使わない多角形の書き方と，多角形を利用したプログラムを2つ紹介します．但し，まだ勉強していない三角関数を使っていますから，アルゴリズムを理解できなくても無理ありません．簡単に説明しますと，半径Rの円の円周をN分割（三角形ならば3分割）して頂点の座標を求め，それぞれを直線で結んでいます．

三角関数については，順を追って解説しますから，後でもう一度アルゴリズムを見直してください．

リスト10

```
100 '-- EXAMPLE (1)  --
110  SCREEN 0,3:CLS 3:SCREEN 0,0
120  PI=3.14159
130  ASP=.5:A1=0:A2=2*PI
140  XO=320:YO=100
150  FOR N=3 TO 9
160    R=105+N*10:CL=N-2
170    GOSUB *POL
180  NEXT
190 END
200 '--  polygon  --
210 *POL
220  INC=(A2-A1)/N             ←――― 円周をN分割する
230  FOR TH=A1 TO A2+.1 STEP INC
```

```
240     X=R*SIN(TH)+XO
250     Y=R*COS(TH)*ASP+YO
260     IF TH=A1 THEN POINT(X,Y)
                        ELSE LINE -(X,Y),CL
270  NEXT
280 RETURN
```

リスト11

```
100 '-- EXAMPLE (2)  --
110  SCREEN 0,3:CLS 3:SCREEN 0,0
120  PI=3.14159
130  ASP=.5:A1=0:A2=2*PI
140  XO=320:YO=100
150  N=12:R=180:CL=3
160  GOSUB *POL
170 END
180 '--  diamond  --
190 *POL
200  INC=(A2-A1)/N
210  FOR TH1=A1 TO A2-.1 STEP INC
220    FOR TH2=0 TO TH1 STEP INC
230      X1=R*SIN(TH1)+XO
240      Y1=R*COS(TH1)*ASP+YO
250      X2=R*SIN(TH2)+XO
260      Y2=R*COS(TH2)*ASP+YO
270      LINE(X1,Y1)-(X2,Y2),CL
280    NEXT
290  NEXT
300 RETURN
```

ここで述べた例以外にも，さまざまな図形の描き方があります．図形を描く時は，そのときどきに応じて，最も適したプログラムを考えてみてください．ひとつの図形を描くにも，プログラムによって，かかる時間が違ってきます．

2.3 円を描く

円を描く命令は，最近のパーソナル・コンピュータには必ず備わっています．もちろん，本書が対象としている N_{88}-BASIC にも備わっています．しかし，円を描く命令が備わっていない機種もありますし，また，直線の時と同様に円を描くアルゴリズムを知りたい方もいるでしょう．そこで，ここではそのアルゴリズムからお話しすることにします．もちろん，いちはやく円を描く命令を知りたい方は，2.3.1を読み飛ばして2.3.2から読んでも構いません．

2.3.1 円を描くには

紙に円を描く道具としては，コンパスがあります．中心と半径を決めて，くるっとコンパスを回せば円が描けます．ところが，円をCRT上に描こうとする時は，コンパスが使えません．そこで，このコンパスに変わるものを考えることにしましょう．

まず，「円」とは何でしょうか．数字の教科書をひもとくと，「円とは，平面上の定点から等距離にある点の軌跡をいい，この定点を円の中心，定点と軌跡の距離を半径という」というのが載っています．

この文から円の方程式を求めて，それを使って円を描くことにします．図17(a)のように，中心

O（x_0, y_0）と点P（x, y）の距離rは，ピタゴラスの定理により，

$$r=\sqrt{(x-x_0)^2+(y-y_0)^2} \quad \cdots\cdots(14)$$

となります．ここで簡単にするため，図17(b)のように中心Oが原点にあると仮定します．この時(14)式は，

$$r=\sqrt{x^2+y^2} \quad \cdots\cdots(15)$$

となります．右辺と左辺をそれぞれ自乗しても右辺と左辺は等しいので，(15)式の両辺をそれぞれ自乗すると，学校で習った円の方程式，

$$r^2=x^2+y^2 \quad \cdots\cdots(16)$$

となります．この式からx，yを求める式を導きますと，次式のようになります．

$$\begin{cases} x=\pm\sqrt{r^2-y^2} \\ y=\pm\sqrt{r^2-x^2} \end{cases} \quad \cdots\cdots(17)$$

この(17)式を満たす円は図18のようになると考えられます．この図から，xは－rからrまで，yは－rからrまでの範囲をとることが分かります．ですから，無闇にx，yの値を変えて(17)式を満たすx，yの組（点）を探すより，xまたはyのどちらか一方を－rからrまで変化させて(17)式よりx，yの組（点）を求めた方が，簡単にしかも早く円が描けます．

このような考え方を用いたプログラムがリスト12です．リスト12では，xの値を－rからrまで変化させて，(17)の下の式でyの値を求めて，円を描いています．この時，注意してもらいたいのが，グラフィック座標系への変換と画面の縦・横（アスペクト）比です．グラフィック座標系への変換は，1070行と1080行でXO，YOをそれぞれX，Yに加えることで行っています．縦・横比は，640×400の白黒高分解能モードでは1：1ですが，640×200のモードでは，横100ドットと縦50ドットは画面上で同じ長さを表すことを意味しています．リスト12では，この縦・横比の補

図17　円の方程式の考え方

図18　円の方程式の軌跡

正を1070行と1080行でYの値に ASP をかけることで行っています．640×400の白黒高分解能モードの場合は，120行を，

```
120 ASP=1:C=7:R=80
```

と変更してください．

リスト12

```
100 '--  Draw circle 1  --
110  SCREEN 0,0:CLS 3
120  ASP=.5:C=7:R=80
130  XO=160:YO=110
140  GOSUB *CIR1
150 END
1000 '
1010 '   ｴﾝ ﾉ ﾎｳﾃｲｼｷ ｦ ﾂｶｳ
1020 '      X*X+Y*Y=R*R
1030 '
1040 *CIR1
1050  FOR X=-R TO R
1060    Y=SQR(R*R-X*X)
1070    PSET(X+XO,Y*ASP+YO),C
1080    PSET(X+XO,-Y*ASP+YO),C
1090  NEXT
1100 RETURN
```

リスト12で描いた円は，円の左側（x＝－r付近）と，円の右側（x＝r付近）の時と円の上下（x＝0付近）の時では，ドットの間隔が違います．(16)式を使っても，工夫を加えれば，ドット連続するように描かせることができますが，これはあなたが考えてみてください．

それでは，この方法以外に円を描く方法はないでしょうか．ここで思い着くのは「極座標」です．極座標系は，図19のように，平面上の任意の点（x，y）を原点（0，0）からの距離rとx軸からの仰角θで表す座標系です．図をよく見て理解してください．この極座標（r，θ）から今まで使ってきた直交座標（x，y）に変換するのは実に簡単で次の式によって行われます．

$$\begin{cases} x = r\cos\theta \\ y = r\sin\theta \end{cases} \qquad \cdots\cdots(18)$$

さて，極座標系で表された円の方程式は，

$$\begin{cases} r = 一定 \\ 0 \leqq \theta \leqq 2\pi \end{cases} \qquad \cdots\cdots(19)$$

という式になりますので，これを(18)式に代入すると，

$$\begin{cases} x = r\cos\theta \\ y = r\sin\theta \end{cases} \qquad \cdots\cdots(20)$$

但し，$0 \leqq \theta \leqq 2\pi$，r は一定（半径）

となります．

図19　極座標系

実は，この(20)式は(16)式を変形したにすぎないのです．(16)式の右辺に(20)式を代入してr^2になることを確かめてみましょう．

$$\begin{aligned} x^2+y^2 &= (r\cos\theta)^2+(r\sin\theta)^2 \\ &= r^2\cos^2\theta+r^2\sin^2\theta \\ &= r^2(\cos^2\theta+\sin^2\theta) \end{aligned}$$

$\cos^2\theta+\sin^2\theta=1$になることは分かっていますから，上の式は$r^2$となります．これで，(20)式は確かに(16)式と同じ円を表すことが分かりますね．(20)式以外にも(16)式を操作することによって，

$$\begin{cases} x = 2t/(1+t^2) \\ y = (1-t^2)/(1+t^2) \end{cases} \qquad \cdots\cdots(21)$$

但し，$t = \tan\left(\dfrac{\theta}{2}\right)$

が得られます．これも，x，yを求める式に $t = \tan(\frac{\theta}{2})$ を代入することによって(16)式と同じ式であることが分かりますが，話が長くなりますのでその証明は省きます．

リスト13は(20)式を使った円を描くプログラムです．θの刻みを小さくすれば，ドットが連続します．

リスト13

```
100 '--  Draw circle 2  --
110  SCREEN 0,0:CLS 3
120  ASP=.5:C=7:R=80
130  XO=160:YO=110
140  GOSUB *CIR2
150 END
2000 '
2010 '    sine cosine ヲ ツカウ
2020 '      X= R*cos(t)
2030 '      Y=-R*sin(t)
2040 '
2050 *CIR2
2060  DEG=2*3.14159/360 ←── θの刻みは1度にする
2070  FOR T=0 TO 359
2080    X=R*COS(T*DEG)
2090    Y=R*SIN(T*DEG)
2100    PSET(X+XO,Y*ASP+YO),C
2110  NEXT
2120 RETURN
```

ここで円を描く方法の原点に戻って考えてみましょう．コンパスで描く時は，中心と半径を決めて，くるっと回して書きますね．これと同じ方法で円を描く方式があります．それは「回転移動」を使う方法です．

円の中心を点O，そこから半径rだけ離れた点をPとします．この点Pを，点Oを中心にして回転移動させると，その軌跡は円となります．

図20　回転移動による円のプロット

図20を使って，具体的に説明しましょう．ここで，話を簡単にするため，点Oは原点(0，0)にあると仮定します．図のように最初の点を(r，0)の位置にとります．この点Pを，点Oを中心にθだけ反時計回りに回転移動させると，点P_1 (x_1, y_1)に移ります．さらに，点P_1を同様

に回転移動させると，点P_2（x_2，y_2）に移ります．これを必要な回数だけ繰り返すと，図のように円が描けます．図ではθを30°としたために，「円」と呼べるものではありませんが，θを小さくすればドットの間隔が縮まって「円」に見えます．

この時使う回転移動の式は，点P_n（x_n，y_n）をθ回転移動して得られる点をP_{n+1}（x_{n+1}，y_{n+1}）とすると，

$$\begin{cases} x_{n+1}=x_n\cos\theta-y_n\sin\theta \\ y_{n+1}=x_n\sin\theta+y_n\cos\theta \end{cases} \qquad \cdots\cdots(22)$$

となります（詳細は8章でお話しします）．

この回転移動の式を使って，円を描くプログラムをリスト14に示します．(22)式では$\cos\theta$と$\sin\theta$は定数ですので，3080行で変数ＣＯとＳＩに代入して使っています．また，3100行と3110行で，前の点の座標を必要とするため，変数ＸＮとＹＮを使っていることに注意してください．

リスト14

```
100 '--  Draw circle 3  --
110  SCREEN 0,0:CLS 3
120  ASP=.5:C=7:R=80
130  XO=160:YO=110
140  GOSUB *CIR3
150 END
3000 '
3010 '   ｶｲﾃﾝ ｲﾄﾞｳ ﾉ ｼｷ ｦ ﾂｶｳ
3020 '     X(n+1)=X(n)*cos(t)-Y(n)*sin(t)
3030 '     Y(n+1)=X(n)*sin(t)-Y(n)*cos(t)
3040 '
3050 *CIR3
3060  X=R:Y=0                    ←── X(0),Y(0)
3070  T=2*3.14159/360            ←── Tを1度とする
3080  CO=COS(T):SI=SIN(T)  ← cos(T)とsin(T)は先に
                                    求める
3090  FOR N=0 TO 360
3100    XN=X*CO-Y*SI  ┐
3110    YN=X*SI+Y*CO  ├ 次の点を求める
3120    X=XN:Y=YN     ┘
3130    PSET(X+XO,Y*ASP+YO),C
3140  NEXT
3150 RETURN
```

さて，θを小さくすればドットが連続するようになりますが，さらにθを小さくすると(22)式はどうなるでしょうか．θを0に近づけると，

$$\sin\theta \fallingdotseq \theta, \quad \cos\theta \fallingdotseq 1$$

になることが知られていますので，これを(22)式に代入してみましょう．但し，θが非常に小さいので，その意味を込めてθをεに置き変えています．

$$\begin{cases} x_{n+1}=x_n-\varepsilon\cdot y_n \\ y_{n+1}=\varepsilon\cdot x_n+y_n \end{cases} \qquad \cdots\cdots(23)$$

リスト15

```
100 '--  Draw circle 4  --
110  SCREEN 0,0:CLS 3
120  ASP=.5:C=7:R=80
130  XO=160:YO=110
140  GOSUB *CIR4
150 END
4000 '
4010 '  ｽﾊﾟｲﾗﾙ ｦ ｶｸ
4020 '     X(n+1)=X(n)-E*Y(n)
4030 '     Y(n+1)=E*X(n)+Y(n)
4040 '
4050 *CIR4
4060  X=R:Y=0
4070  E=2^(-4)
4080  X0=R:Y0=Y
4090 *CIR4.1
4100   XN=X-E*Y
4110   YN=Y+E*X
4120   X=XN:Y=YN
4130   PSET(X+XO,Y*ASP+YO),C
4140   IF X-X0<R THEN *CIR4.1 ←── 終了条件の判定
4150 RETURN
```

この式を使ったプログラムがリスト15です．しかし，このプログラムを実行させると，円を描かずに，うずまき（スパイラル）を描いてしまいました．そこで，うまいことを考えた人がいました．それは(23)式でy_{n+1}を求める式のx_nの代わりにx_{n+1}を使うことです．

$$\begin{cases} x_{n+1}=x_n-\varepsilon \cdot y_n \\ y_{n+1}=\varepsilon \cdot x_{n+1}+y_n \end{cases} \quad \cdots\cdots(24)$$

この式を使って円を描くサブルーチンが，リスト16です．このプログラムを実行すると，ちゃんと円が描けるでしょう．

リスト16

```
100 '--  Draw circle 5  --
110  SCREEN 0,0:CLS 3
120  ASP=.5:C=7:R=80
130  XO=160:YO=110
140  GOSUB *CIR5
150 END
4000 '
4010 '  DDA ｦ ﾂｶｳ
4020 '     X(n+1)=X(n)-E*Y(n)
4030 '     Y(n+1)=E*X(n)+Y(n)
4040 '
4050 *CIR5
4060  X=R:Y=0
4070  E=2^(-8)
4080  ENDX=Y*E/2+E:ENDY=X*E/2+E ← 終了条件値の計算
4090  X0=R:Y0=Y
4100 *CIR5.1
4110   X=X-E*Y
4120   Y=Y+E*X
4130   PSET(X+XO,Y*ASP+YO),C
4140   IF ABS(X-X0)>ENDX OR ABS(Y-Y0)>ENDY THEN *CIR5.1 ←── 終了条件の判定
4150 RETURN
```

このプログラムのアルゴリズムは，機械語やハードウェアで実現することができます．リスト16には掛け算や割り算が含まれていますが，ASPは2または1ですから，機械語のシフト命令で計算できます．また，Eを2^{-m}の形にすると，E＊YやE＊Xもシフト命令で計算できます．こ

の時mは，グラフィックの分解能にもよりますが，N_{88}-BASICでは8ぐらいで良好な結果が得られます．

実は，ここで紹介したアルゴリズムが，直線のところでもお話ししたDDAと呼ばれるもので，グラフィックスでの図形の発生に幅広く利用されています．N_{88}-BASICでも，このDDAが使われていると考えられます．そのうえ，さらに高速に描くように工夫されています．もう，お気付きの方もいらっしゃるでしょうが，円を描く場合，円周全体を計算する必要はありません．円は中心を通る任意の直線について線対称なので，実際は図21の太線の部分を求めるだけで充分なのです．

図21

本章ではアルゴリズムや考え方に重点を置いたので，プログラムはBASICで書いたものだけを載せました．そこで，円を描くことを機械語などで実現することは，読者の皆さんの課題としておきます．

2.3.2 円を描く命令CIRCLE文

N_{88}-BASICで円を描く命令は，CIRCLE（サークル）文です．CIRCLE文は，

CIRCLE（x，y），r，色，開始角度，終了角度，比率

という形式で使います．（x，y）は円の中心の位置で，グラフィック座標（ワールド座標）で扱い，rは半径の大きさで，比率を指定しない時は水平方向の半径で指定します．色は〝パレット番号〟によって指定し，描く円の色を決めます．もしこれが省略されたら，COLOR文によって設定されている，グラフィック画面のフォア・グラウンド・カラーが採用されます．命令実行後，LPは円の中心の座標に設定されます．

```
circle(100,60),80          ←─白で左上に円が描かれる
Ok
circle(300,60),100,4 ←─緑でその右に少し大きい円が描かれる
Ok
```

円の中心の座標の指定には，PSET などと同じように相対座標も使えます．

```
circle step(200,90),50 ←─ 画面の右下に円が描かれる
Ok
```

今までのように，開始角度や終了角度を指定しないと円を描きますが，開始角度や終了角度を指定すると，指定された角度の範囲内のみに円孤を描きます．開始角度や終了角度の値と実際の位置との関係は図22のようになっていて，開始角度や終了角度はラジアンで指定します．この時注意してもらいたいのは，開始角度と終了角度の値と，実際に描かれる円孤の位置の関係です．円孤は，開始角度で指定された位置から反時計回りに終了角度の位置まで描くからです．次の例で，このことを確かめてください（但し，$\pi/2 \fallingdotseq 1.572$，$3\cdot\pi/2 \fallingdotseq 4.7124$）．

図22　開始角度と終了角度の値と実際の位置の関係

```
circle(100,60),80,1,1.571,4.7124←─青で左半分の円弧を書く
Ok
circle(300,60),80,2,4.7124,1.571←─ 赤で右半分の円弧を書く
Ok
```

開始角度や終了角度が負であった場合には，その絶対値をとり，正にした値によって円孤が描かれますが，その時中心から半径が描かれます．この機能を使うと簡単に扇形を描くことができます．

```
circle(100,60),100,4,-1.571,-3.142 ←─ 緑で扇形を描く
Ok
```

開始角度，終了角度は，－2πから＋2πまでの範囲にないと〝Illegal Function Call〞（?FC Error）のエラーになります．

```
circle(100,60),100,,-8,10
Illegal function call        −2π〜＋2πの範囲を越えているためエラーが生じた
Ok
```

今までの実行例は真円でしたが，比率を指定することによって楕円を描くこともできます．比率は，(垂直方向の半径)/(水平方向の半径)で指定します．640×200ドットのモードは0.5，640×400ドットのモードでは1.0を指定すると，真円を描きます．比率を指定した時半径の指定に注意が必要です．比率が1以上の場合には，半径は垂直方向の半径を意味します．次の例で確かめてください（640×200モードの時）．

```
circle(320,100),80,4,,,1  ←─縦長の楕円を描く
Ok
circle(320,100),80,1,,,.5 ←─縁の楕円に左右が接した真円を描く
Ok
circle(320,100),80,2,,,2  ←─縁の楕円に上下が接した楕円を描く
Ok
```

CIRCLE命令のサンプルとして，リスト17のようなプログラムを作りました．このプログラムは，動物の角を表現したつもりなのですが，いかがでしょうか．

```
100 '--   EXAMPLE (for 400 mode)  --
110  SCREEN 2,0:CLS 3:CONSOLE ,,0,1
120  ASP=1:C=4:R=2:OX=420:OY=100
130  COLOR@(0,0)-(79,24),4
140  GOSUB *CIR2
150 END
160 '
170 *CIR2
180  DEG=2*3.14159/360
190  FOR T=0 TO 360*2.5-1 STEP 15
200    X=R*COS(T*DEG)
210    Y=R*SIN(T*DEG)
220    CIRCLE(X+OX,Y*ASP+OY),R/2,4
230    R=R+R^1.1/16
240  NEXT
250 RETURN
```

リスト17

2.4 図形に色を塗る

今まで描いてきた図形は線で描いた図形でした．唯一違っていたのはBFオプション付きのLINE文で，この命令で描かれた長方形はその中が色で塗りつぶされています．しかし，これ以外の図形──例えば平行四辺形に色を塗る時などには，この命令は使えません．そこで，N_{88}-BASICでは，任意の図形に色を塗るために，PAINT命令が用意されています．ここではこのPAINT命令を使って，面的な図形を描いてみましょう．

PAINT文は，

PAINT (x, y), 塗る色, 境界色

という形式で使います. (x, y) は, 塗り始める位置で, グラフィック座標 (ワールド座標) で扱います. 塗る色, 境界色は共にパレット番号で指定し, 境界色で囲まれた領域を〝塗る色〟で塗りつぶします. ですから, 塗りたい領域は必ず閉じていなければなりません. もし1ドットでも開いている所があれば漏れ出してしまいます. それともう1つ, PAINT 命令で指定する座標は, 現在のビューポートに現れているワールド座標の値しか受け付けません. もし, ビューポートの外の座標を指定すると〝Illegal Function Call (?FC Error)〟となります.

以上のことに気を付ければ, 別段難しいことではありません. 今まで描いてきた図形などに簡単に色付けすることができます.

まずダイレクトモードで実行してみましょう.

```
circle(320,100),80,7 ←― 中央に白で円を描く
Ok
paint(320,100),7,7   ←― 円を白く塗りつぶす
Ok
paint(320,100),0,0   ←― 円を黒く塗り変える
Ok
circle(320,100),80,4 ←― 緑で円を描く
Ok
paint(320,100),2,4   ←― 円の中を赤で塗りつぶす
Ok
```

それでは次に2.2.3の多角形の所で解説した五角形のプログラム (リスト9) を使って簡単なプログラムを作ってみましょう (リスト18). このプログラムで色付けをしているのは, 180行であり, 五角形の頂点の座標から2ドット下の座標を塗り始めの位置として指定しています. 塗る色は五角形を描いた色と同じであり, ループにより青(1)から白(7)まで変化しています.

リスト18

```
100 '--  PAINT SAMPLE (1)  --
110  SCREEN 0,0:CLS 3
120  N=200
130  FOR C=1 TO 7
140    X=50+(C-1)*50
150    Y=(199-N/2)-C*10
160    POINT(X,Y):RESTORE
170    GOSUB *PENT
180    PAINT STEP(0,2),C,C
190  NEXT              ← 五角形の中をカラーCでペイント
200 END
500 '--  Pentagon  --
510 *PENT
520  FOR I=0 TO 4
530    READ X(I),Y(I)
540  NEXT
550  POINT STEP(N*X(0),N*Y(0))
560  X(5)=POINT(0):Y(5)=POINT(1)
570  FOR I=1 TO 4
580    LINE -STEP(N*X(I),N*Y(I)),C
590  NEXT
600  LINE -(X(5),Y(5)),C
610 RETURN
620 DATA .476,0,-.476,.173,.155,.280
630 DATA .697,0,.155,-.280
```

(500〜630行) リスト9と同じ正五角形を描くプログラム

では次に，もう少し複雑な図形に色付けしてみましょう．基本図形は三角関数を用いて描き，菱形の部分をペイントします．PAINT 命令は，先程も説明した通り，完全に閉じた図形の内側の点を指定しなければなりませんから，このように狭い領域をペイントする場合，シヴィアーな位置指定が要求されます．

PAINT 命令には，この他にタイル・ストリングを使ってタイル・パターンをペイントする機能などがありますが，その機能は 4 章のグラフィック・ツールの紹介でお話しすることにして，ここでは基本図形に 8 色をペイントすることにとどめておきます．

リスト19

```
100 '--  PAINT EXAMPLE (2)  --
110  SCREEN 0,3:CLS 3:SCREEN ,0
120  DEF FNX(X,Y)=SIN(X)*Y+OX        } 円周上の座標を求める関数
130  DEF FNY(X,Y)=COS(X)*Y*ASP+OY    }
140  PI=3.14159:ASP=.5
150 '---< draw line >---
160  N=50:R2=180:R1=R2/3
170  INC=2*PI/N:D=2*PI/8
180  OX=320:OY=100:C=7
190  GOSUB *FRAME
200 '---< painting >---
210  FOR I=1 TO 5
220    READ AS:PC=I
230    R3=R2*AS:GOSUB *PAINTCL
240  NEXT
250 END
260 DATA .83,.64,.51,.44,.39
270 '
280 *FRAME '---< draw frame >---
290  FOR TH=0 TO PI*2 STEP INC
300    X1=FNX(TH,R1)  :Y1=FNY(TH,R1)
310    X2=FNX(TH,R2)  :Y2=FNY(TH,R2)
320    X3=FNX(TH+D,R1):Y3=FNY(TH+D,R1)
330    X4=FNX(TH-D,R1):Y4=FNY(TH-D,R1)
340    LINE(OX,OY)-(X1,Y1),6
350    LINE(X3,Y3)-(X2,Y2),C
360    LINE        -(X4,Y4),C
370  NEXT
380 RETURN
390 '
400 *PAINTCL '---< paint color >---
410  FOR TH=SANG TO SANG+PI*2 STEP INC
420    PX=FNX(TH,R3):PY=FNY(TH,R3)   } 菱形の中の座標を求めてカラーPCでペイント
430    PAINT(PX,PY),PC,C             }
440  NEXT
450 RETURN
```

3章　任意の関数をプロットする

3.1　任意の関数をプロットするには

前章では，座標とはどのようなものであるか，またいかにして点を打ったり，線を引いたりするかということについて述べてきました．だいたい前章までで，点や線の描き方は理解できたことと思います．

この章では，前章の理解をさらに深め，この応用として，数学的要素を含めながら話を進めていきます．コンピュータ・グラフィックスを行う上で，数学という学問は大変に重要であり，グラフィックスの基礎となります．しっかりと要点を押さえるようにしてください．しかし，数学とはいっても，難しいものは必要ではなく，中学校もしくは高校の初等程度で十分ですから，気楽に考えてくださって結構です．また，忘れてしまった人は，中学校の数学の本などを参考にしながら，読んでいってください．

それでは，徐々に本題に入っていきたいと思います．

3.1.1　ワールド座標とウィンドウ

座標という概念を取り入れると，前章で述べたように任意の位置を定めるのに，大変効果的であることは分かると思います．コンピュータには，この座標が大きく分けてキャラクタ座標系とグラフィック座標系の2つがあります．しかしグラフィックスを行う場合，分解能の点でキャラクタ座標は，大変不利であるため，一般に美しい曲線や細かい図形を描けるグラフィック座標を用いることになります．

N_{88}-BASICにおいては，このグラフィック座標が，さらに2つの座標系に分かれています．その1つをワールド座標と呼び，もう1つをスクリーン座標と呼びます．ここでは，N_{88}-BASICでグラフィックスを行う上で大変重要である，ワールド座標系を中心にお話ししていきます．このワールド座標は，元来の画面上で固定されてしまった座標とは違い，画面の外にも座標が広がっているという考えを取り入れた座標なのです．

つまり，このワールド座標は，架空の座標であり，大変に大きな値を持っている座標なのです．

図1 世界地図

図2 ①の枠で囲った場合

図3 ②の枠で囲った場合

では何故，このように架空（論理的）の座標が必要なのでしょうか．その最大の理由は，この概念を取り入れるとグラフィックスを行う上で，大変効果的であるということです．

では，どのように効果的であるかということを，具体的に説明します．

まず，図1のような世界地図を思い浮かべてみてください．今，一定の大きさの紙面上に，①と②の枠で囲った部分を，図2，図3のように引き伸したとします．この2つの図の関係は，図2を元に考えますと，図3は拡大図にあたります．また，図3を元に考えますと，図2は縮小図にあたる訳です．つまり，図1のような大きな世界地図において，枠の囲む大きさによって拡大，縮小が自由自在になる訳です．

これは一例にすぎませんが，他にも写真などを写すときにもこれと同様なことがいえます．このことを頭に入れて，もう一度，ワールド座標について考えてみましょう．

大変大きな世界地図を，ワールド座標に置き換えて考えてみますと，先程の作業がワールド座標上でも実現できることになります．つまり，ある図やグラフをワールド座標上に描きますと，拡大・縮小を思いのままに操ることができるようになります．このようなことから，ワールド座標という概念が取り入れられていることが理解できるでしょう．

では，ワールド座標をどのように用いていけばよいのでしょうか．

ワールド座標は，大変大きな座標ですから，その中から見たい領域を指定して，その領域を画面に表示するようにします．N_{88}-BASIC では，そのような動作を行う命令を〝WINDOW〟命令と呼びます．WINDOW 命令の詳しい説明は，後述，実例を示しながら説明しますので，ここでは以下のように覚えておいてください．

〝WINDOW〟命令を用いると，大きなワールド座標の中の任意の領域を抽出することができる．

また，点を打つ命令の〝PSET〟や線を引く命令の〝LINE〟などは，ワールド座標上で実行されますので，このことも頭の隅にでも置いておいてください．

では，もう1つの座標系，すなわち，スクリーン座標にも少し触れておきましょう．スクリーン座標とは，ディスプレイ画面上のある領域内に存在する固定された（物理的な）座標です．この画面上の，ある領域のことを〝ビューポート〟と呼び，ビューポートはディスプレイ画面上で見ることのできる領域なのです．つまり図4のようになっています．このビューポートを指定する命令を〝VIEW〟命令と呼びます．この〝VIEW〟命令も後述します．

図4

3.1.2 方程式を描くには

先程数学は，コンピュータ・グラフィックスの基礎であると述べましたが，数学の中で特に重要なのが〝方程式〟です．この方程式のグラフを，画面に表示できるようになればしめたものです．これが簡単に実現できるようになると，方程式を与えられた時にも，何も困ることはありません．その方程式を，コンピュータを用いて自動的に画面に表示させることにより，方程式の内容を容易に理解することができます．

このことはどういうことかというと，物理などを例にとると，ボールはどのように飛んでいくか，振り子はどのように動くかなどが画面でシミュレートすることによって，理解できるのです．これはまた，ゲームなどにも数多く応用できます．では方程式のグラフを，いかにしたら画面に描くことができるでしょうか．

方程式とは，直線や曲線を表す式であり，x軸（水平軸）とy軸（垂直軸）との関係を表すものです．つまり，x軸の値を決定すると，y軸の値は必然的に決まります．このとき，xとyの組ができるために，ある位置が決定します．このxとyの組がすなわち座標にあたるわけです．

y
点の集まりである
0
x

図5

この点をいくつも求めれば，直線や曲線が描けます(図5)．もう少し詳しくいいますと，ある方程式のグラフを正確に描きたいときには，xの値を細かくとっていけばいいことになります．しかし，コンピュータで描く場合には，ディスプレイ画面の解像度の問題がどうしても絡んできてしまい，あまり細かな値を取っても意味がなくなってしまいます．

今の話から予想がつくと思いますが，画面上にグラフを描くには，ある方程式にいくつかのxの値を代入して，いくつかの（x，y）の組を求めます．そして，この（x，y）の組を画面上に順々にプロットすれば，直線や曲線のグラフが得られるのです．しかし，わたしたちがxの値を次々に変えていくのでは大変手間がかかってしまいます．そこで，プログラムでループを作り，値を順次変えていきます．ループの中で代表的なものといえば，やはり，"FOR～TO～STEP……NEXT" でしょう．この章ではこの命令を用いて，いろいろな関数をプロットしていきます．では，どのようにプログラムを構成するのかを説明しましょう．

まず，どのような方程式のグラフを描くかを決めたら，xの値をいくつからいくつまで変化させるのかを決めます．つまりxの値の範囲を定めるのです．次に，その方程式をFOR～NEXTループの中に置きます．これでループの中でxの値が変化するごとに，yの値が計算され(x，y)の組が次々と発生していきます．そして，その都度（x，y）の組をプロットしていけば，直線や曲線が描ける訳です．

```
FOR  x＝（始まりの値）  TO  （終りの値）  STEP（ステップ幅）
  ☆ 方程式  （例えば  y＝ax＋b  など）
  ☆（x，y）の組をプロット  （PSET）
NEXT
```

以上に述べたことが，方程式を描くための一番大切な基礎となります．ここで，今までのことをまとめてみましょう．

(1) 方程式y＝f（x）を求める．

(2) xの範囲を決める．この時，xの取りうる範囲を把握しておかなければなりません．

(3)　xに対してyの範囲がどのようになるか，一応把握する．

(4)　方程式をループの中に入れて，プロットする．この時，ステップを決めなければなりません．

方程式のプロットのおおまかな方法は，だいたい理解できたと思います．

では，ここで重要と思われる方程式の一般形を用いて並べてみます．

① 1次関数　$y=ax+b$

② 2次関数　$y=ax^2+bx+c$

③ 高次関数　$y=a_nx^n+a_{n-1}x^{n-1}+\cdots\cdots+a_0x^0$

④ ルート関数　$y=\sqrt{ax+b}$

⑤ 対数関数　$y=a\log x$

⑥ 指数関数　$y=a^x$

⑦ 三角関数　$y=\sin x$　：$y=\cos x$　：$y=\tan x$

⑧ 円　$x^2+y^2=r^2$

ここまでで，方程式のグラフを描くための準備は，だいたい備いました．では，この後は実際にプログラムを示しながら説明をしていきます．

まず初めは，簡単な1次関数である直線の方程式を考えてみましょう．直線については，2章ですでに学んでいますが，ここではグラフという意味で捕えています．

3.2　1次関数

1次関数とは，直線の方程式で，xとyの関係は一般に，

$y=ax+b$　（a，bは任意の定数）

の形で表されます．

それでは，例としてy＝xのグラフを描いてみましょう．このときxの範囲は，いろいろな値が取れますが，ここでは$-100\leqq x\leqq 100$としましょう．

このy＝xのグラフを描くには,先程お話しした手順でプログラムを組んでいけばよいのです．まずこの方程式y＝xをFOR～TO…NEXTループの中に入れます．この時,変数名はxであり，yの値は，xの変化範囲を－100から100までと決めましたので,－100から100となります．またステップは，今回1としておきます．これでxが変化するごとに（x，y）の組が発生してきますので，PSETを用いて次々とプロットします．プログラムの一例としてリスト1を挙げておきます．グラフィックスを扱う場合このプログラムの初めのように,イニシャライズ（初期設定）をする癖をつけておいてください．

では，このプログラムを実行してみてください．どうなりましたか．写真のような直線が描けたと思います．

リスト1

```
100 '
110 ' y=x
120 '
130 SCREEN 0,3:CLS 3:SCREEN,0:COLOR 7,0,0,7 ⎫
140 CONSOLE 0,25,0,1:WIDTH 80,25            ⎭ 初期設定
150 '
160 ' plot
170 '
180 FOR X=-100 TO 100
190   Y=X
200   PSET (X,Y),4 ←── 点のプロット
210 NEXT
220 '
230 END
```

しかし，このグラフは少しおかしい所があるのです．というのは私たちが知っているy＝xのグラフは，図6のようなグラフです．何故，このような違いが生じたのでしょうか．

この違いの大きな原因は，座標に問題があります．〝PSET〞命令は，始めにお話ししたワールド座標上の任意の位置にドットをセットします．そして，このワールド座標は，初期状態（WINDOWを用いていない場合）では，（図7）のように設定されます．ですから，先程のプログラムを実行しますと（図8）のような表示になってしまいます．

図6

図7

図8

これを解決するには，2つの方法があります．

まず第1には，今のx軸とy軸の座標系を新しいグラフィック座標であるX軸とY軸に変換する方法です．第2の方法としては，現在取り扱っている座標系はワールド座標ですから，〝WINDOW〞命令に注目して，画面に表示するワールド座標の領域を，見易いように切り直す方法です．

では，第1の方法から説明していきましょう．現在の座標は，前の図に示したように，左上が原点となっておりx軸が右方向に，またy軸が下方向に延びています．先程のグラフの違いを取り除くには，この座標の原点を，ディスプレイ画面の中心に移動した図9のような新しいグラフィック座標（x軸，y軸）に変換する必要があります．この時，元の座標系の任意の点（x，y）と新しい座標系の任意の点（X，Y）の関係は，以下のようになります．

$$X = x + 320$$
$$Y = 100 - y$$

この関係を利用してプログラムを書き直してみましょう．書き直す場所は，1箇所でリスト1の200行目を，

```
PSET（x＋320，100－y），4
```

とすれば完了です．

図9

しかし，この方法は，ワールド座標というものをまるで無視した方法であり，N_{88}-BASICにおいては，よい方法とはいえません．私としては，この方法はあまり感心できません．やはり，次に述べる第2の方法をとって頂きたいと思います．

この方法は，N_{88}-BASICの優れた座標系，すなわちワールド座標に注目し，これをフルに活用しようではないかということです．はじめにお話ししたように，ワールド座標を用いると，表示領域の取り方によって，拡大や縮小が自由にできます．さらに，グラフなどの一部を表示したり，全体を表示したり，私たちの思い通りにグラフをアレンジすることができます．そして，このグラフをアレンジするときに使用する命令が〝WINDOW〞命令です．この命令で，ある表示領域を指定することを一般に〝ウィンドウを切る〞といいます．今後，このような言葉が出てきた場合

図10

図11

は，この意味を言っているんだなあと思ってください．

まず，先程のプログラム（リスト1）を実行すると，現在ワールド座標上では，図10のようになっています．では，私たちがよく知っている図6のようにするには，どうしたらよいのでしょう．

これは，実に簡単です．現在の画面表示領域を図11のように，変えればよいのです．つまり，ワールド座標をWINDOW命令によって〝xは−100から100まで〟また〝yも−100から100まで〟切ればよい訳です．これで，y＝x（$-100 \leqq x \leqq 100$）の全体のグラフが，画面全体に描けるようになります．このとき，画面上の座標は，図12のようになっています．

もう1つ気を付けなければならないことがあります．それは，y軸の正（＋）の方向が下向きに取ってあることです．私たちがよく用いるのはyの正の方向は上向きですから，ここでも上向きに取りましょう．これは，yの符号を逆に，すなわちyに〝−〟を付ければよいことになります．実際には，右辺の式の値の正負を逆にすればよいのです（$y=f(x) \rightarrow y=-f(x)$）．

以上のことをまとめてリスト1を書き直してみましょう．

```
WINDOW (−100, −100)− (100, 100)
```

を〝plot〟の前に挿入し，190行を，

図12

Y＝－X

に書き直せばよいのです．

では，実行してみてください．

あとは，〝y＝2x＋5〟などの関数も今までの応用ですから，いろいろと試してみてください．また，ウィンドウの切り方もいろいろ変えてみると面白いと思います．

例えば，

WINDOW（－1000，－1000）－（1000，1000）

とすると，先程のグラフは縮小されます．

実際に，ウィンドウでいろいろ遊んでみると，ワールド座標というものがどのようなものか，あるいは，ワールド座標という考え方の素晴しさがよく分かると思います．

では，次に2次関数あるいは，3次関数などの高次関数を描いてみましょう．

3.3　2次関数（高次関数）

2次関数は，曲線の方程式では代表的なものであり，特に，放物線と呼ばれています．放物線とは，字の通りで，物を投げ上げた時に描く軌跡のことです．この関数は，一般に$y=ax^2+bx+c$（$a\neq 0$，b，cは任意の定数）という形で表されます．

では，最も一般的な例として，$y=x^2$を扱ってみましょう．xのとりうる範囲は，$(-\infty \leqq x \leqq \infty)$ですが，今回はxの範囲を$(-100 \leqq x \leqq 100)$とします．まず$y=x^2$という方程式をFOR～NEXTの中に入れ，xを－100から100まで変化させこのとき発生した（x，y）の組をPSETを用いてプロットします．STEP幅は1にしておきます．座標は先程と同様に，ワールド座標をWINDOWによって，(－100，－100)－(100，100)で切ります．以上をまとめたプログラムがリスト2です．実は，先程のプログラムと，ほとんど変わりません．

Y＝－Xが，Y＝－（X＊X）になっただけですね．この〝－〟は，前と同じくyの正の方向を上方向にするためです．では，実行させてみてください．

どうですか．写真のような放物線のグラフが描けたでしょう．

リスト2

```
100 '
110 ' y=x*x
120 '
130 SCREEN 0,3:CLS 3:SCREEN,0:COLOR 7,0,0,7
140 CONSOLE 0,25,0,1:WIDTH 80,25
150 '
160 ' window
170 '
180 WINDOW (-100,-100)-(100,100)
190 '
200 ' plot
210 '
220 FOR X=-100 TO 100
230   Y=-(X*X)    ←─Y軸を逆にする
```

```
240   PSET (X,Y),4
250 NEXT
260 '
270 END
```

今回は，もう少しグラフらしく，x軸，y軸を引いて，表示する画面も正方形にしてみましょう．x軸やy軸などの線を引くには，〝LINE〞命令を用います．現在の座標は，図12のようになっていますから，x軸は〝y＝0〞，y軸は〝x＝0〞の直線です．つまり，x軸は，

LINE（−100, 0）−（100, 0）

y軸は，

LINE（0，−100）−（0，100）

となります．この2行を，plotの前あたりに挿入すればよいわけです．

次に，少しグラフを見易くするために，画面の表示領域を少し変えてみましょう．

今までは画面全体を用いた，x軸方向に延びた長方形でしたね．これを，図13のように正方形に変えます．これは，前にお話ししたビューポートを少し変えることによって可能です．つまり，画面をグラフ用紙にたとえたならば，グラフ用紙をいっぱいに使用するか，あるいは，グラフ用紙の一部を使用するかというのと同じことです．

現在のビューポートを図13のようにするには，まず，縦と横の大きさを決めなければなりません．今，y軸（縦）方向はいっぱいに使い，200としましょう．x軸（横）方向は，正方形ですから200ですね，とするのは誤りです．というのは縦･横比の問題でx軸とy軸の大きさの比率が違うからです（640×400モードの時に限り1：1）．つまりx軸方向は，2倍して400となります．この時，ビューポートの枠を書くために上下1ドットを削ります．

次に，ビューポートをどこに置くかということですが，やはり，中央にするのが見易いでしょう．

〝VIEW〞命令を使用して以上のことをまとめてみると，

VIEW（100, 1）−（500, 198),, 3

となります．これを，WINDOWの前に挿入して実行してください．写真のようになるはずです．

図13

ここで，もう少しウィンドウとビューポートを変えて，少し変わったことをしてみましょう．写真を見てください．これは，画面上に3つのビューポートを切り，左側には，$y=x^2$（$-100\leqq x\leqq100$）のグラフの全体を，右側には，その一部を表示させたものです．リスト3を見てもらえば分かると思いますが，ビューポードとウィンドウをうまく組み合わせるだけで，N_{88}-BASICではこのようなことが容易に実現できます．

リスト3

```
100 '
110 ' y=x*x    (view & window)
120 '
130 SCREEN 0,3:CLS 3:SCREEN,0:COLOR 7,0,0,7
140 CONSOLE 0,25,0,1:WIDTH 80,25
150 '
160 ' ----- MAIN -----
170 '
180 VIEW (10,10)-(400,190),,3
190 WINDOW (-100,-10000)-(100,100)
200 GOSUB *XYLINE
210 GOSUB *PLOT
220 '
230 VIEW (420,10)-(620,100),,5
240 WINDOW (0,-1000)-(50,0)
250 GOSUB *XYLINE
260 GOSUB *PLOT
270 '
280 VIEW (420,105)-(620,195),,6
290 WINDOW (-10,-100)-(0,0)
300 GOSUB *XYLINE
310 GOSUB *PLOT
320 END
330 '
340 ' x,y line
350 '
360 *XYLINE
370  LINE (-100,0)-(100,0)
380  LINE (0,-10000)-(0,100)
390 RETURN
400 '
410 ' plot
420 '
430 *PLOT
440  FOR X=-100 TO 100
450    Y=-(X*X)
460    PSET (X,Y),4
470  NEXT
480 RETURN
```

今までのグラフは，すべて点のプロットによって描いていた訳ですが，今度は，実線によりグラフを描いてみましょう．これも，ちょっとした工夫で実現できます．

今回は，少し傾きの滑かな$y=x^2/50$（$-100\leqq x\leqq100$）を扱ってみましょう．

実線を引くのに一番単純な方法は，やはりFOR～NEXTループのSTEPの数を細かく取る方法でしょう．しかし，この方法を使うと大変時間がかかります．さらにこの方法は，見かけ上，点と点がつながっているように見えるだけです．従って，ウィンドウを小さく切ってしまうと，アラがでてきてしまいます．そこで他の方法を考えてみます．

他の方法で実線を引くとしたら，やはり〝LINE〟命令を用いるのが一番よいでしょう．この〝LINE〟を用いる時に，少し工夫をします．N_{88}-BASICの特徴の一つであるLP(Last referenced Point)を積極的に利用するのです．この〝LP〟というのは，グラフィック命令で座標を与えた時に，最

後に指定された座標を覚えています．

つまり，〝LP〟を考慮して LINE 命令を行いますと，図14のように現在の〝LP〟から，新しく指定した任意の点まで線で結んでくれます．ですから始めに〝POINT〟命令で〝LP〟を指定しておき，あとはFOR～NEXT で任意の点を発生させ，その都度 LINE 命令で線を引けば，実線で放物線などが描けるわけです．では，実際にプログラムを書いてみましょう．

LP
新しく指定した点
線を引く

図14

一例として，リスト 4 を挙げておきます．このプログラムの330行から360行が，今お話ししたことを実現した部分です．これを少し説明してみましょう．FOR～NEXT はもうお分かりですね．340行の〝－〟は，先程と同じように，y の正方向を上方向に変換するものです．そして，350行が今回のプログラムのメインとなる部分です．このグラフは，x＝－100からx＝100まで変化するのですから，始めの点はx＝－100です．この時，先程の〝LP〟を指定します．つまり，LP は，x＝－100，Y＝－（100×100／50）＝－200　となるわけです．あとは，その〝LP〟から LINE 命令によって次々に発生してくる点を結んで放物線を描くわけです．

このプログラムの始めの部分も少し説明しましょう．ウィンドウは，先程と同じにしておきます．ビューポートは，グラフの上にメッセージを入れるために，先程のものよりy方向を10削りました．あとは，x軸とy軸を引き，x，y，0と文字を入れるようにしています．

リスト4

```
100 '
110 ' y=x*x/50
120 '
130 SCREEN 0,3:CLS 3:SCREEN,0:COLOR 7,0,0,7
140 CONSOLE 0,25,0,1:WIDTH 80,25
150 LOCATE 28,0:PRINT "y=x*x/50"
160 '
170 ' window & view
180 '
190 VIEW (100,10)-(500,198),,3
200 WINDOW (-100,-100)-(100,100)
210 '
220 ' x,y
230 '
240 LINE (-100,0)-(100,0)
250 LINE (0,-100)-(0,100)
260 LOCATE 60,13:PRINT "x"
270 LOCATE 36,2:PRINT "y"
280 LOCATE 36,13:PRINT"0"
290 LOCATE 0,0
300 '
310 ' line
320 '
330 FOR X=-100 TO 100
340   Y=-(X*X/50)
350   IF X=-100 THEN POINT (X,Y) ELSE LINE-(X,Y),4
360 NEXT
370 '
380 END
```

（240～290行：x軸，y軸を表示）
（350行：各点の間を線で結ぶ）

では，このプログラムを，実行してみてください．今回は，だいぶグラフらしく描かれたと思います．今回も，グラフの全体を描いていません．グラフ全体をみる時は，先程と同様にウィンドウを切り直します．

例えば，

WINDOW（－100, －200)－（100, 100)

のようにします．

3次関数（$y=ax^3+bx^2+cx+d$）や4次関数など高次関数のグラフの描き方も，今までお話ししてきた1次関数や，2次関数とまったく同様の考え方でいいのです．ただ1つだけ気を付けなければならない点は，次数が高くなるほど，値が爆発的に大きくなることです．従ってグラフの全体を見るには，ウィンドウを大きく切らなければならなくなります．

3.4 ルート関数と分数関数

3.4.1 ルート関数（無理関数）

ルート関数は無理関数であり，関数の中にルートが含まれるものをいいます．これは，一般形で表せば，$y=\sqrt{ax+b}$ という形で表せます．

N_{88}-BASICには，〝SQR〟と呼ばれるルート関数が定義されていますから，ルート関数のグラフを描いたり，またルートを計算するには，この〝SQR関数〟を用いることができます．

ルート関数は，今までの1次関数や2次関数のようにxの定義域を無制限に取ることはできません．皆さんもご承知の通り，ルートの中は，負でない数しか取れないのです．ですから，xは0または正の数のみを取るわけです．つまり，xの定義域は〝$x \geqq 0$〟となります．また当然ですが，yの値も〝$y \geqq 0$〟となります．もし，xの定義域を〝負〟にして，〝SQR関数〟にxの値を代入すると，〝ERROR〟が発生します．

では，以上のことに注意して，ルート関数のグラフを描いてみましょう．

今回は，例として，$y=7\sqrt{x}$ のグラフを描きます．これは，一般形の〝$a=49$, $b=0$〟の場合に当るわけです．xの値は，先程お話しした通り，〝$x \geqq 0$〟ですから，ここではxの定義域を〝$0 \leqq x \leqq 100$〟とします．

また，今回のSQR関数は，ウィンドウを必ず切らなければなりません．というのは，このままでプログラムを組んで実行させると，画面に何も出力されないのです．この理由は，図15を見て頂ければ理解できると思います．そこでウィンドウの切り方は，xとyの範囲を考慮して，WINDOW（－100, －100)－（100, 100）とします．ビューポートなどの設定は，2次関数の時と同じにしておきます．

図15

以上のことをまとめて，実際にプログラムを組んだのがリスト5です．

リスト5

```
100 '
110 ' y=sqr(x)
120 '
130 SCREEN 0,3:CLS 3:SCREEN,0:COLOR 7,0,0,7
140 CONSOLE 0,25,0,1:WIDTH 80,25
150 LOCATE 33,0:PRINT "y=7*sqr(x)"
160 '
170 ' view & window
180 '
190 VIEW(100,10)-(500,198),,3
200 WINDOW(-100,-100)-(100,100)
210 '
220 ' x,y
230 '
240 LINE(-100,0)-(100,0)
250 LINE(0,-100)-(0,100)
260 LOCATE 60,13:PRINT"x"
270 LOCATE 36,2:PRINT"y"
280 LOCATE 36,13:PRINT"0"
290 LOCATE 0,0
300 '
310 ' plot
320 '
330 FOR X=0 TO 100
340   Y=-(7*SQR(X))
350   PSET(X,Y),4
360 NEXT
370 '
380 END
```


では，実行させてみてください．

写真のようなグラフが描けたと思います．このように，放物線の半分をねかせたものが，ルート関数のグラフなのです．リスト5は2次関数のときとほとんど変わりませんので，説明は省略します．

また，ルート関数はちょっとした特徴を持っています．それは，2次関数と深い関わり合いがあるのです．このルート関数は2次関数の逆関数なのです．つまり先程少し触れましたが，放物線を90°回転させた一部分なのです．例としてあげた$y=7\sqrt{x}$は，$y=\frac{1}{49}x^2$の逆関数という訳です．ですから，$y=7\sqrt{x}$のグラフの上に，$y=\frac{1}{49}x^2$（$x\geqq 0$）のグラフを描くと，$y=x$で対称なグラフになります．

このように，画面上でいろいろなグラフを描いてみますと，関数同志がどのような関係にあるかが，一目で分かります．これも，グラフィックスの特徴であるといえましょう．

3.4.2 分数関数

ここでは，簡単な分数関数を扱います．分数関数は，一般に $y=(cx+d)/(ax+b)$ （$a\neq0$，$ad-bc\neq0$）という形で表されます．分数関数のグラフを描く時に，一番注意しなければならないことは，無理関数と同様にxの定義域です．分数において，分母が〝0〞になるとyの値は無限大になってしまうために，分母を0にするようなxの値は定義域から除かなければなりません．このことに気を付ければ，後は無理関数と同じ考え方でよい訳です．

では，最も簡単な形をしている $y=100/x$ のグラフを描いてみましょう．xの定義域は，$-100\leqq x\leqq100$ としましょう．但し，先程お話ししたように，分母が0になるようなxは除外します．つまり $x\neq0$ の領域です．除外の仕方としては，IF～THEN 文を用いるのがよいでしょう．つまり，$x=0$ の時は，計算もプロットもしないように，飛ばしてしまえばよいのです．

ウィンドウの切り方は，xとyの範囲を考慮して，

WINDOW (−100, −100)− (100, 100)

とします．VIEW などの条件は，今まで通りです．

リスト6はそのプログラムの一例です．今回のプログラムは，先程のプログラムと基本的な考え方は同じですが，一部違うことに気が付くと思います．このプログラムでは，オーバフローの検出の仕方に多少工夫がしてあります．オーバフローは，今回の例では，$x=0$ の時に起こるのですが，STEP 幅を細かく取りますと，$x=0$ 以外の場合にも，起こることがあります．この理由は，内部計算の誤差で，STEP幅を細かく取ったために，$x=0$ 付近の値が，$x=0$ に非常に近い値になってしまうからです．この問題を解決するには，リスト6の350行目のように，$x=0$ 付近の値を除外するようにします．他にも，yの値でオーバフローを検出する方法がありますが，今回は，xの値による方法を用いました．

リスト6

```
100 '
110 ' y=100/x
120 '
130 SCREEN 0,3:CLS 3:SCREEN,0:COLOR 7,0,0,7
140 CONSOLE 0,25,0,1:WIDTH 80,25
150 LOCATE 33,0:PRINT"y=100/x"
160 INC=1 ←―― STEP数
170 '
180 ' view & window
190 '
200 VIEW(100,10)-(500,198),,3
210 WINDOW(-100,-100)-(100,100)
220 '
230 ' x,y
240 '
250 LINE(-100,0)-(100,0)
260 LINE(0,-100)-(0,100)
270 LOCATE 60,13:PRINT"x"
280 LOCATE 36,2:PRINT"y"
290 LOCATE 36,13:PRINT"0"
300 LOCATE 0,0
310 '
320 ' plot
330 '
340 FOR X=-100 TO 100 STEP INC
350   IF ABS(X)<INC THEN *JUMP ←― オーバフローの判定
```

```
360    Y=-100/X
370    PSET (X,Y),4
380 *JUMP:NEXT
390 '
400 END
```

さらに，このグラフを実線で描いてみたり，ウインドウを切り直して，x軸やy軸付近の変化の仕方などを研究してみてください．但し，実線で描く時には，多少注意しなければなりません．実線で描く場合には，今まで通りLINE命令を用いるのですが，今回のグラフは不連続なグラフですから，実行中にLPを指定し直さなければなりません．

そのことを考慮したのがリスト7です．先程のリスト6のplotの部分と入れ変えてください．このプログラムは，yの値がある程度大きくなった時に，LPを指定し直して実線を描かないようにしてあります．これにより，2つの曲線はつながらないで描けます．

ではこれからは，少し変わった特徴を持つ関数を扱っていきます．まず最初は，対数関数のグラフの描き方について説明していきましょう．

リスト7

```
310 '
320 ' plot
330 '
340 FOR X=-100 TO 100 STEP INC
350   IF ABS(X)<INC THEN *JUMP
360   Y=-100/X
370   IF (Y>150) OR (Y<-150) THEN POINT (X,Y) : GOTO *JUMP
380   IF X=-100 THEN POINT (X,Y) ELSE LINE-(X,Y),4
390 *JUMP:NEXT
400 '
410 END
```

3.5 対数関数

対数関数は一般に，$y=a\log_b x$ という形で表されます．この式の中のbのことを〝底〞と呼びます．この底は，一般に〝10〞と〝e〞の2つを用います．10を底とした対数を〝常用対数〞と呼び，e（2.71828…）を底とした対数を〝自然対数〞と呼びます．しかし一般にコンピュータで対数を使用する時は，自然対数の方を用いることになっています．今後は，単に対数関数といったらeを底とした自然対数を示していると思ってください．また，N_{88}-BASICにおいて，この対

数関数は〝LOG″として定義されており，この〝LOG″関数の底もやはり〝e″です．

対数関数も無理関数と同じように，xの定義域に問題があります．対数関数において，xの定義域は〝正″の範囲しかとれません．つまり〝x＞0″です．

では，この点に注意してy＝3logxのグラフを描いてみましょう．今までは，xの定義域を大きくとっていましたので，今回は0＜x≦5としてみましょう．この時念を押しておきますが，xは0を含みません．

1つに，x＝0から始めず，xより大きな値，例えば，0.5や1などから始める方法があります．しかし，今回はそのような方法を用いずに，x＝5からプロットを始めて，x＝0になったら実行を中止するようにします．これはIF～THENを用います．Disk-BASICの方は，WHILE～WENDを使えばさらに簡単になります．またxが，5から0と範囲が小さいために，STEPの数が1では，プロットする数が少なすぎますからSTEP　－0.05とします．

ウィンドウの切り方は，xとyの範囲を考慮して，WINDOW（－5，－10）－（5，10）とします．以上のことをまとめてプログラミングしたのがリスト8です．

リスト8

```
100 '
110 ' y=log(x)
120 '
130 SCREEN 0,3:CLS 3:SCREEN,0:COLOR 7,0,0,7
140 CONSOLE 0,25,0,1:WIDTH 80,25
150 LOCATE 33,0:PRINT "y=3*log(x)"
160 '
170 ' view & window
180 '
190 VIEW(100,10)-(500,198),,3
200 WINDOW(-5,-10)-(5,10)
210 '
220 ' x,y
230 '
240 LINE(-5,0)-(5,0)
250 LINE(0,-10)-(0,10)
260 LOCATE 60,13:PRINT"x"
270 LOCATE 36,2:PRINT"y"
280 LOCATE 36,13:PRINT"0"
290 LOCATE 0,0
300 '
310 ' plot
320 '
330 FOR X=5 TO 0 STEP -.05
340   IF X=<0 THEN END     ' x>0 ←― 終了条件の判定
350   Y=-(3*LOG(X))
360   PSET(X,Y),4
370 NEXT
380 '
390 END
```

このグラフをながめてみますと，対数関数はy軸に限りなく近づいていくことが分かると思います．もし，y軸付近をもう少し詳しく知りたい場合は，xの正の部分を拡大してみてください．これはウィンドウを切り直すことによって実現できます．この場合，STEPの幅をもう少し小さく取った方が，美しい曲線が描けると思います．

また，対数関数のグラフは，いろいろ描いてみると分かると思いますが，yの値は，単調に増加するのです．ですから，このようなグラフは0付近を描くことに意味があり，xの範囲を大き

くしても，あまり意味がありません．このように，グラフの特徴を摑んでおくこともなかなか重要で，美しいグラフや模様が描けるようになります．

3.6 指数関数

指数関数は，一般に $y=a^x$ ($a>0$, $a\neq 1$) という形で表します．そして，このaのことを対数関数の時と同じように〝底〟と呼びます．この指数関数は，先程の対数関数と深い関係があります．というのは，先程の対数関数は，この指数関数の逆関数なのです．ですから対数関数の時と同じように，底の値（aの値）を〝e〟とします．底の値を〝e〟とした時の一般形は，$y=e^x$，または $y=\exp(x)$ という形で表されます．N_{88}-BASICでは，底を〝e〟とした指数関数が定義してあり，〝EXP（x）〟という形で用います．

この指数関数を用いる時は，xの定義域はどのような値でもとれますから対数関数の時のように定義域に気遣う必要はありません．

例として，$y=\exp(x)$　（$-10\leqq x\leqq 10$）を扱います．

プログラムについては，今までとほとんど同じですから，結果だけ載せておきます(リスト９)．但し，このプログラムは，指数関数を実線で描くようにしてあります．また，逆関数である対数関数との関係が摑めるように，対数関数も一緒にプロットするようにしました．

リスト9

```
100 '
110 '  y=exp(x)
120 '
130 SCREEN 0,3:CLS 3:SCREEN,0
140 CONSOLE 0,25,0,1:LOCATE 33,0:PRINT"y=exp(x)"
150 '
160 '  view & window
170 '
180 VIEW(100,10)-(500,198),,3
190 WINDOW(-10,-10)-(10,10)
200 '
210 '  x,y
220 '
230 LINE(-10,0)-(10,0)
240 LINE(0,-10)-(0,10)
250 LOCATE 60,13:PRINT"x"
260 LOCATE 36,2:PRINT"y"
270 LOCATE 36,13:PRINT"0"
280 LOCATE 0,0
290 '
300 '  line draw  [y=exp(x)]
310 '
320 FOR X=-10 TO 10 STEP .1
330   Y=-(EXP(X))
340   IF Y<-10 THEN *EX
350   IF X=-10 THEN POINT (X,Y) ELSE LINE-(X,Y),4
360 NEXT
370 '
380 '  y=x , y=log(x)
390 '
400 *EX
410   FOR X=-10 TO 10 STEP .5
420     Y=-X
430     PSET (X,Y)
440   NEXT
450 '
460 *EX1
```

```
470    FOR X=10 TO 0 STEP -.5
480      IF X=<0 THEN END
490      Y=-(LOG(X))
500      PSET (X,Y),2
510    NEXT
520 '
530 END
```

このグラフを眺めてみますと，指数関数と対数関数はｙ＝ｘに関して対称であることがよく分かります．

では次に，図形処理において，最も重要な三角関数についてお話しします．

3.7 三角関数

三角関数は，グラフィックスを行う上で頻繁に用いられます．というのは，この三角関数が，図形を回転させたり，移動させたりするのに都合がいいからなのです．ですから，今までの関数とは，多少，性質や取り扱い方に異なった面があります．またこの三角関数は，この章において大変重要な部分ですから，しっかりポイントを押えるようにしてください．もし，分からないことがでてきたら，専門書などを開いてみてください．

一般に，

$y = \sin x$,　$y = \cos x$,　$y = \tan x$

の３つをまとめて三角関数と呼びます．この三角関数の変数ｘは，今までの関数とは少し違い，〝度〟や〝弧度（ラジアン）〟で扱わなければならないので注意が必要です．また，三角関数は，一定の範囲において同じ変化を繰り返す特徴を持っています．この範囲のことを〝周期〟といい，このような関数のことを〝周期関数〟と呼びます．

ではまず初めに，三角関数の中で代表的な〝$y = \sin x$〟のグラフについてお話ししましょう．

3.7.1 y＝sinx（正弦関数）

正弦関数は，一般に〝サイン〟と呼ばれます．サイン関数は先程お話ししたように，〝角度の関数〟です．角度の表し方には，〝度〟と〝弧度〟の２つの方法がありますが，コンピュータで角度を扱う場合には，一般に〝弧度法〟を用いることになっています．この弧度法の単位は〝ラジアン〟であり，みなさんもよく知っている〝π〟を用いた角度の表し方です．ですからもし角度が〝度〟で与えられた時には，〝ラジアン〟に変換しなければなりません．この変換の仕方は後で説明しますから，今は〝弧度法〟について話を進めていきます．

N_{88}-BASICにおいてサイン関数は，〝SIN（ｘ）〟として定義されています．ｘの単位は先程お話しした通り，〝ラジアン〟で扱います．ですからグラフを描く時には，〝ラジアン〟を意識しながらプログラムを作るようにします．

今回は例として，$y = \sin x$ のグラフを描いてみます．ｘの定義域は，$(-\pi \leqq x \leqq \pi)$とします．

サイン関数は，〝2π〃を周期とする周期関数ですから，この場合は，サイン関数のグラフを1周期描くことを意味しています．基本的な考え方は，今まで通りで問題ありません．サイン関数をFOR～NEXTの中に入れます．そして，次々と発生してくる点でプロットしていけばよい訳です．xの値は，$-\pi$からπまで変化させる訳ですが，今回πの値は，3.14にしておきます．STEPの数は，1ではプロット数が少なくなってしまいますので，0.1にします．ウィンドウの切り方は，xの範囲が$-3.14 \leqq x \leqq 3.14$，またyの範囲が$-1 \leqq y \leqq 1$ですから，

WINDOW（−3.14，−1）−（3.14，1）

とします．ここで重要なことは，ウィンドウは整数に限らず実数でも構わないということです．ビューポートの方は，今までと同じように切ることにします．あとは，x軸とy軸を引き，y軸の正の方向を上向きに変換しましょう．以上の点を考慮してプログラムしたのが，リスト10です．πの値は，イニシャライズの時に定義しておいた方が，見易くなります．

では，このプログラムを実行させてみてください．写真のようなグラフが描けたと思います．このグラフを見ると，やはり今までのグラフとはかなり違うことが分かるでしょう．

リスト10

```
100 '
110 ' y=sin(x) (RAD)
120 '
130 SCREEN 0,3:CLS 3:SCREEN,0:COLOR 7,0,0,7
140 CONSOLE 0,25,0,1:WIDTH 80,25
150 LOCATE 33,0:PRINT"y=sin(x)"
160 PAI=3.14
170 '
180 ' view & window
190 '
200 VIEW(100,10)-(500,198),,3
210 WINDOW(-3.14,-1)-(3.14,1)
220 '
230 ' x,y
240 '
250 LINE(-3.14,0)-(3.14,0)
260 LINE(0,-1)-(0,1)
270 LOCATE 60,13:PRINT"x"
280 LOCATE 36,2:PRINT"y"
290 LOCATE 36,13:PRINT"0"
300 LOCATE 0,0
310 '
320 ' plot
330 '
340 FOR X=-PAI TO PAI STEP .1 ←― Xを－πからπまで変化させる
350   Y=-SIN(X)
360   PSET(X,Y),4
370 NEXT
380 '
390 END
```

描いたグラフが，サイン関数の1周期（基本周期）です．もし，xの範囲が定まっていなければ，同じようなグラフが左右に広がることになります．さらにこの関数を知るために，ウィンドウを切り直して，最大付近や0付近を拡大してみてください．

では次に，xが〝度〃で与えられた場合について，お話ししましょう．角度の単位を〝度〃から〝ラジアン〃に変換する方法は，別に難しいことではありません．〝度〃と〝ラジアン〃の関係

は，360°＝2πですから，

(ラジアン)＝(度)×π／180

の関係が成り立ち，ラジアンを求めることができます．後は先程とまったく同じです．例として y＝sinx（−360°≦x≦360°）のグラフを描いてみます．今回は，2周期描くことになりますね．yは先程と同じでよいのですが，xは，今回，定義域を−360から360にし，ウィンドウもそれに合わせて切ります．この時，x軸は度で取っていますので，間違わないようにしてください．ラジアンに直すのは，あくまでも計算の時だけです．また，STEP 数は，今回4にしておきます．その他は，今までと同じです．プログラムは，先程のリスト10の一部分をリストのように変えて実行させてみてください．写真のようなグラフが，描けたと思います．

実線で描く場合は，放物線の時のように LINE 命令を用いてください．この時，STEP 幅を大きめに取ると実行速度は向上します．

リスト11

```
100 '
110 ' y=sin(x) (DEG)
120 '
130 SCREEN 0,3:CLS 3:SCREEN,0:COLOR 7,0,0,7
140 CONSOLE 0,25,0,1:WIDTH 80,25
150 LOCATE 33,0:PRINT"y=sin(x)"
160 PAI=3.14
170 '
180 ' view & window
190 '
200 VIEW(100,10)-(500,198),,3
210 WINDOW(-360,-1)-(360,1)
220 '
230 ' x,y
240 '
250 LINE(-360,0)-(360,0)
260 LINE(0,-1)-(0,1)
270 LOCATE 60,13:PRINT"x"
280 LOCATE 36,2:PRINT"y"
290 LOCATE 36,13:PRINT"0"
300 LOCATE 0,0
310 '
320 ' plot
330 '
340 FOR X=-360 TO 360 STEP 4
350   XX=X*PAI/180  'RAD ←— Xをラジアンに変換
360   Y=-SIN(XX)
370   PSET(X,Y),4
380 NEXT
390 '
400 END
```

3.7.2 y＝cosx（余弦関数）

余弦関数は，一般に〝コサイン〟と呼ばれます．このコサイン関数は，サイン関数とよく似た特徴を持っており，サイン関数と同様に〝2π〟を周期とした周期関数で，最大値は〝1〟，最小値は〝−1〟を取ります．また，角度の単位も〝ラジアン〟で扱わなければなりません．ですから，〝度〟で与えられた場合には，やはり〝ラジアン〟に変換します．N_{88}-BASIC では，このコサイン関数は，〝COS（x）〟として定義されています．

コサインのグラフを描くプログラムは，先程のサインのリスト10，リスト11と全く同じです．但し，もちろん SIN を COS に書き変えなければなりませんが．

グラフを見て頂ければ分かると思いますが，曲線の形はサイン関数の時とまったく同じです．このコサイン関数の上に，サイン関数のグラフを描いてみますと，一層はっきりとしますが，このコサイン関数のグラフは，サイン関数のグラフを左へ90度（$\pi/2$）だけ平行移動したものなのです．

3.7.3 y＝tanx（正接関数）

正接関数は，一般に〝タンジェント〟と呼ばれています．このタンジェント関数は，サイン関数やコサイン関数と少し性質が違います．タンジェント関数は周期関数ですが，周期が今までと違い〝π〟なのです．また，タンジェント関数のグラフは，今までのような連続する曲線ではなく，yの値が〝$-\infty$〟から〝$+\infty$〟まで変化するグラフです．ですから，このグラフは不連続なグラフとなります．このことより，タンジェント関数を扱う時は，多少注意を払う必要があります．N_{88}-BASICには，このタンジェント関数は，〝TAN（x）〟と定義されています．

タンジェント関数を用いる時に，xは角度ですので，サイン関数などと同様に，〝ラジアン〟で扱わなければなりません．今回は，$y=5\tan x$　（$-180<x<180$）のグラフを実線で描いてみます．

しかし，ここで注意しなければならないことは，サインなどとは違ってxが，±90°の時に無限大となってしまい，オーバフローを起こしてしまうことです．このエラーを起こさないように何かしら手を打たなければなりません．このエラーをトラップする方法は，いくつかありますが，その1つは3．4．2の分数関数のところで紹介しましたので，今回は，ON ERROR GOTO 命令を使用する方法を用います．この方法を使うと，オーバフローなどのエラーが生じた場合に，エラー処理ルーチンに制御が移り，なんらかの回復処理を行った後，実行を再開させることができます．従って，この方法を用いれば実行が中断されることがなく処理速度が向上し，プログラムが簡潔になります．これを実際にプログラム上で実現したものが，リスト12です．

また，タンジェント関数は，不連続な関数ですから，分数関数の時と同じように，LPの再指定が必要です．ウィンドウは，yの範囲が無限大であるため，適当な値で切っておきます．今回は，－100から100としました．では，リスト12を実行させてみてください．写真のようなグラフが描け

たと思います．このグラフを見ると，確かに，±90°の時に無限大になっています．

リスト12

```
100 '
110 ' y=tan(x) (line draw)
120 '
130 ON ERROR GOTO *ERRORTRAP ←— エラー処理ルーチンの定義
140 '
150 SCREEN 0,3:CLS 3:SCREEN,0
160 CONSOLE 0,25,0,1:WIDTH 80,25
170 LOCATE 33,0:PRINT"y=tan(x)"
180 PAI=3.14159
190 '
200 ' view & window
210 '
220 VIEW(100,10)-(500,198),,3
230 WINDOW(-180,-100)-(180,100)
240 '
250 ' x,y
260 '
270 LINE(-180,0)-(180,0)
280 LINE(0,-100)-(0,100)
290 LOCATE 60,13:PRINT"x"
300 LOCATE 36,2:PRINT"y"
310 LOCATE 36,13:PRINT"0"
320 LOCATE 0,0
330 '
340 ' plot
350 '
360 FOR X=-180 TO 180
370   XX=X*PAI/180
380   Y=-(5*TAN(XX))
390   IF (Y<-200) OR (Y>200) THEN POINT (X,Y):GOTO *JUMP
400   IF X=-180 THEN POINT (X,Y) ELSE LINE-(X,Y),4
410 *JUMP:NEXT
420 END
430 '
440 *ERRORTRAP
450   RESUME *JUMP ←— エラー処理ルーチン
```

今回の関数のようにエラーが起こる場合には，ON ERROR GOTO は大変都合のよい命令です．みなさんもこのようなエラーが生じる場合は，この命令をどんどん活用してみてください．

3.7.4 サインとコサインを合成する

今までは，三角関数を個別的に扱ってきました．そこで今回は，今まで説明してきました三角関数を用いて，ちょっとした応用をしてみようと思います．

まず初めに，サイン関数とコサイン関数を合成すると，どのようなグラフになるのかやってみましょう．

サイン関数とコサイン関数の合成というと，難しく聞えるかもしれませんが，簡単にいえば2つの関数の〝和〟をとるだけです．ですから，なにも難しいことはありません．

例として，　$y=100\,(\sin x+\cos x)$ $(-\pi \leqq x \leqq \pi)$ のグラフを描いてみます．この方程式もxの値が定まれば，yの値も定まってしまいます．ですから，考え方はサイン関数の時と同じです．但し，STEP 幅は，今までと違う決定の仕方をします．今までのように適当な値を扱ってもよいのですが，場合によっては内部計算の誤差で，最後の値がくるってしまい，グラフが見にくくなることもあります．ですから今回は，プロット数を指定して，xの範囲をこのプロット数で割り，

今までの方法	E* ステップ幅	計算誤差のためにE*の部分で実行を終了してしまう.
プロット数を指定する方法	E	xの範囲を等間隔で分けると，きちんとEの部分で実行が終了する.

図16

xの範囲を等間隔にしてプロットさせるようにします（図16）.

プロット数を90とすると，

(3.14－（－3.14))／90

となります．この結果が，今回の STEP 幅になる訳です．STEP 幅を決定するには，このような方法もあることを覚えておいてください．

また，ウィンドウを切る時に，yの範囲に気を付けなければなりません．サインとコサインを合成した時の，最大値は1より大きくなり，結果として100を越えてしまいます．ですから多少大きめに切ることにします．

プログラムの一例としてリスト13を挙げておきます．このプログラムは，個々の関数と合成した関数の関係が摑めるように，サイン関数とコサイン関数は点線で描き，合成した関数は実線で描くようにしてあります．

リスト13

```
100 '
110 ' y=sin(x)+cos(x)   (RAD)
120 '
130 SCREEN 0,3:CLS 3:SCREEN,0:COLOR 7,0,0,7
140 CONSOLE 0,25,0,1:WIDTH 80,25
150 LOCATE 28,0:PRINT"y=100*(sin(x)+cos(x))"
160 PAI=3.14
170 INC=PAI*2/90 ←― ステップ幅の計算
180 '
190 ' view & window
200 '
210 VIEW(100,10)-(500,198),,3
220 WINDOW(-3.14,-150)-(3.14,150)
230 '
240 ' x,y
250 '
260 LINE(-3.14,0)-(3.14,0)
270 LINE(0,-150)-(0,150)
280 LOCATE 60,13:PRINT"x"
290 LOCATE 36,2:PRINT"y"
300 LOCATE 36,13:PRINT"0"
310 LOCATE 0,0
320 '
330 ' plot
340 '
350 FOR X=-PAI TO PAI STEP INC
360   PSET (X,-100*SIN(X)),1 '--- y=sin(x)
370   PSET (X,-100*COS(X)),5 '--- y=cos(x)
380 NEXT
390 '
400 ' line
```

```
410
420 FOR X=-PAI TO PAI STEP INC
430   Y=-(100*(SIN(X)+COS(X))) '--- y=sin(x)+cos(x)
440   IF X=-PAI THEN POINT (X,Y) ELSE LINE-(X,Y),4
450 NEXT
460 '
470 END
```

他にもいろいろな合成が考えられますが，今までのちょっとした応用ですから，いくらでも描くことができると思います．サインを自乗にしたり位相をずらしてみたりいろいろとやってみるとおもしろいと思います．

次に今までと少し変わった，媒介変数で表された方程式について説明します．

3.8 媒介変数を用いた曲線

今まで扱ってきた方程式は，(x，y)の方程式でしたが，今回は，xとyをtなどの関数として扱う媒介変数（パラメータ）表示についてお話しします．

この媒介変数を用いる方法は，一般によく使われる方法です．というのは，(x，y)の方程式では大変複雑な方程式であっても，この媒介変数を用いることによって，多くの場合簡単な方程式で表すことができるようになるからです．また，任意の点を表すのに，yがxの関数であるよりは，yとxがある媒介変数の関数で求められる方が都合がよい場合があります．例えば，物理の例をとってみましょう．球を投げた時に，球の任意の位置を決定するのには，yがxの関数であるよりは，yとxが時間の関数である方がより一般的で扱い易くなります．さらに，場合によっては，yとxがある媒介変数で表せるが，yがxの関数であるように書き直すのが難しい場合もあります．以上のように，媒介変数を用いた方が，方程式が扱い易い場合が多いのです．

但し，媒介変数を用いる場合に，何が何の関数であるかということを押さえておかないと，混乱する元となってしまいます．

では初めに例として，

$$x = 1 + t^2,\quad y = 2 - t - t^2\ (-5 \leqq t \leqq 5)$$

のグラフを描いてみましょう．

この関数のグラフを描く時に注意しなければならないことは，xとyは，tの値が定まると決定することです．他の事については，今まで通りの考えでいいのです．

まず，この2つの関数をFOR～NEXTの中に入れ，tを－5から5まで変化させます．ウィンドウは，今までと少し違う切り方をします．これは，グラフの都合上見易いようにするためです．ビューポートなどについては，今まで通りです．では，プログラムをリスト14に掲げておきます．実行させてみてください．写真のようなグラフが描けたことと思います．

この2つの関数は，tを消去することによって，(x，y)の方程式に直すことができます．この方程式は，少し複雑な$x^2+2xy+y^2-7x-6y+10=0$となります．このことから，(x，y)

の方程式では，複雑なものであっても，今回のように媒介変数を用いると，簡単な方程式で扱うことができることが分かります．

リスト14

```
100 '
110 '  x=1+t*t  y=2-t-t*t
120 '
130 SCREEN 0,3:CLS 3:SCREEN,0:COLOR 7,0,0,7
140 CONSOLE 0,25,0,1:WIDTH 80,25
150 LOCATE 28,0:PRINT"x=1+t*t  y=2-t-t*t"
160 '
170 '  view & window
180 '
190 VIEW(100,10)-(500,198),,3
200 WINDOW(-5,-5)-(26,28)
210 '
220 '  x,y
230 '
240 LINE(-5,0)-(26,0)
250 LINE(0,-5)-(0,28)
260 LOCATE 60,5:PRINT "x"
270 LOCATE 20,23:PRINT "y"
280 LOCATE 20,5:PRINT "0"
290 LOCATE 0,0
300 '
310 '  plot
320 '
330 FOR T=-5 TO 5 STEP .05
340   X=1+T*T
350   Y=-(2-T-T*T)
360   PSET(X,Y),4
370 NEXT
380 '
390 END
```

では次に，この媒介変数を用いた典型的な曲線の例を2つ紹介しましょう．

第1の例として，アステロイド（星芒形）と呼ばれる曲線を描いてみます．この曲線は，星のような形をしており，式では一般に，$x^{\frac{2}{3}}+y^{\frac{2}{3}}=a^{\frac{2}{3}}$（$a>0$）という形で表されます．この曲線を描くには，このままでは描きにくいために，媒介変数を用いて表すようにします．この表し方は，

$$x=a\cos^3\theta \quad y=a\sin^3\theta$$

となります．ですから，アステロイドを描く場合には，xとyをθの関数として扱えばよいのです．

今回は，$x=\cos^3 t$　$y=\sin^3 t$ のグラフを描いてみます．この時，θはtで表して，tは0から2πに制限しておきます．ウィンドウは，x，yともに－1から＋1まで切ります．これは，一般形においてx，yともに最大値a，最小値－aを取るためです．ビューポートなどは，今まで通りです．

プログラムをリスト15に挙げておきます．このプログラムでは，曲線をLINE命令を用いて，実線で描くようにしてあります．この曲線は星形をしており，今までと少し違うことに気付くと思います．このアステロイドは，円を円周に沿ってころがした時に，ころがる円周上の一点が描く軌跡を表しています．

リスト15

```
100 '
110 ' asteroid
120 '
130 SCREEN 0,3:CLS 3:SCREEN,0:COLOR 7,0,0,7
140 CONSOLE 0,25,0,1:WIDTH 80,25
150 LOCATE 33,0:PRINT "asteroid"
160 PAI=3.14159
170 INC=PAI*2/90
180 '
190 ' view & window
200 '
210 VIEW(100,10)-(500,198),,3
220 WINDOW(-1,-1)-(1,1)
230 '
240 ' x,y
250 '
260 LINE(-1,0)-(1,0)
270 LINE(0,-1)-(0,1)
280 LOCATE 60,13:PRINT"x"
290 LOCATE 36,2:PRINT"y"
300 LOCATE 36,13:PRINT"0"
310 LOCATE 0,0
320 '
330 ' line
340 '
350 FOR T=0 TO 2*PAI STEP INC
360   X=COS(T)*COS(T)*COS(T)
370   Y=-SIN(T)*SIN(T)*SIN(T)
380   IF T=0 THEN POINT(X,Y) ELSE LINE-(X,Y),4
390 NEXT
400 '
410 END
```

第2章で学んだ三角関数を使った円の描き方も，この方法と同じ媒介変数によるものだったのです．

では次に，もう1つの例として，リサージュ曲線を描いてみましょう．このリサージュ曲線は，2つの単振動を合成した場合の軌跡を表すもので，一般に，

$$x = A\cos(w_1 t + \alpha)$$
$$y = B\cos(w_2 t + \beta)$$

という形で表されます．

今回は，$x=30\sin 4t$，$y=20\sin 5t$　$(0 \leqq t \leqq 2\pi)$ のグラフを描いてみます．STEP幅は，プロット数を今回180とし，PAI＊2／180とします．ウィンドウは，$-A \leqq x \leqq A$，$-B \leqq y \leqq B$ となりますから，xは，−30から30，yは−20から20で切ります．

リスト16

```
100 '
110 ' lissajous
120 '
130 SCREEN 0,3:CLS 3:SCREEN,0:COLOR 7,0,0,7
140 CONSOLE 0,25,0,1:WIDTH 80,25
150 LOCATE 33,0:PRINT "lissajous"
160 PAI=3.14159
170 INC=PAI*2/180
180 '
190 ' view & window
200 '
210 VIEW (100,10)-(500,198),,3
220 WINDOW (-30,-20)-(30,20)
```

```
230 '
240 ' x,y
250 '
260 LINE (-30,0)-(30,0)
270 LINE (0,-20)-(0,20)
280 LOCATE 60,13:PRINT "x"
290 LOCATE 36,2:PRINT "y"
300 LOCATE 36,13:PRINT "0"
310 LOCATE 0,0
320 '
330 ' line
340 '
350 FOR T=0 TO 2*PAI STEP INC
360   X=30*SIN(4*T)
370   Y=-(20*SIN(5*T))
380   IF T=0 THEN POINT (X,Y) ELSE LINE-(X,Y),4
390 NEXT
400 '
410 END
```

この曲線を見てみると，大変に特殊な曲線であることが分かると思います．また，このグラフは，w_1とw_2の比や α と β の値を変化させることにより，グラフがいろいろと変化します．

今回のアステロイドやリサージュ曲線は，私たちがグラフを書く時は大変苦労しますが，コンピュータはたやすく描いてしまいます．このように，コンピュータを用いて複雑なグラフを描くことは，いろいろな面において重要です．またいろいろな関数を使うことにより，美しい模様や図形を計算によって作り出すことができます．

それでは，この章のまとめとして関数を使った美しいサンプルをいくつか紹介しておきましょう．

但し，リスト17は640×400のモードを使います．

リスト17

```
100 SCREEN 0,3:CLS 3:SCREEN 2,1
110 CONSOLE 0,25,0,1:WIDTH 80,25
120 '
130 WINDOW (-.5,-.5)-(.5,.5)
140 FOR X=-1 TO 1 STEP .02
150   FOR Y=-1 TO 1 STEP .02
160     Z=X*X+Y*Y
170     XX=X/(Z+1):YY=Y/(Z+1)
180     PSET (XX,YY)
190   NEXT
200 NEXT
210 '
220 END
```

```
100 SCREEN 0,3:CLS 3:SCREEN 0,0:COLOR 7,0,0,7
110 CONSOLE 0,25,0,1:WIDTH 80,25
120 '
130 WINDOW (-1,-1)-(1,1)
140 FOR X=-1 TO 1 STEP .05
150   FOR Y=-1 TO 1 STEP .05
160     Z=EXP(-X*X+Y*Y)
170     XX=X/(Z+1):YY=Y/(Z+1)
180     PSET (XX,YY),4
190   NEXT
200 NEXT
210 '
220 END
```

リスト18

リスト19

```
100 ON ERROR GOTO *ERRORTRAP
110 '
120 SCREEN 0,3:CLS 3:SCREEN 0,1:COLOR 7,0,0,7
130 CONSOLE 0,25,0,1:WIDTH 80,25
140 '
150 WINDOW (-4,-4)-(4,4)
160 FOR X=-2 TO 2 STEP .05
170   FOR Y=-2 TO 2 STEP .1
180     Z=X^3-3*X*Y+Y^3
190     IF Z<-1 OR Z>2 THEN 220
200     XX=X/(Z+1):YY=Y/(Z+1)
210     PSET (XX,YY),4
220   NEXT
230 NEXT
240 END
250 '
260 *ERRORTRAP
270   RESUME NEXT
```

次の2つのプログラムは，3次元的に広がる関数ですが，この位になるとかなり見ごたえのあるものが作れます．3次元については，8章で解説します．

リスト20

```
1000 '
1010 ' 3D ?
1020 '
1030  DIM MAX(640),MIN(640),BEFORE(85,1)
1040  P=0 : SCREEN 0,1 : CLS 3
1050 '
1060  FOR I=1 TO 640
1070    MAX(I)=0 : MIN(I)=200
1080  NEXT
1090  FOR I=1 TO 85 STEP 5
1100    BEFORE(I,0)=2*I+394
1110    BEFORE(I,1)=199-I
1120  NEXT
1130 '
1140  FOR Y=2 TO 132 STEP 5
1150    POINT(4-3*Y+400,200-Y/2)
1160    FOR X=1 TO 85 STEP 5
1170      GOSUB *FUNCTION
1180      PY=Z/2+X+Y/2
1190      PX=2*X-3*Y+400
1200      LINE -(PX,200-PY),4
1210      IF Y=2 THEN *SKIP
1220      LINE(BEFORE(X,0),BEFORE(X,1))-(PX,200-PY),4
1230 '
1240     *SKIP
1250      POINT(PX,200-PY)
1260      BEFORE(X,0)=PX:BEFORE(X,1)=200-PY
1270    NEXT
1280  NEXT
1290 END
1300 '
1310 *FUNCTION
1320  S=(X-40)*(X-40):U=ABS(65-Y):T=(U-40)*(U-40)
1330  Z=(1200-S-T)/8
1340  IF Z<0 THEN Z=0
1350  IF Y>65 THEN Z=-Z
1360 RETURN
```

リスト21

```
1000 '
1010 ' 3D ?
1020 '
1030  DIM MAX(640),MIN(640),BEFORE(85,1)
1040  P=0 : SCREEN 0,1 : CLS 3
1050 '
1060  FOR I=1 TO 640
1070    MAX(I)=0 : MIN(I)=200
1080  NEXT
1090  FOR I=1 TO 85 STEP 2
1100    BEFORE(I,0)=2*I+394
1110    BEFORE(I,1)=199-I
1120  NEXT
1130 '
1140  FOR Y=2 TO 132 STEP 2
1150    POINT(4-3*Y+400,200-Y/2)
1160    FOR X=1 TO 85 STEP 2
1170      GOSUB *FUNCTION
1180      PY=Z/2+X+Y/2
1190      PX=2*X-3*Y+400
1200      LINE -(PX,200-PY),2
1210      IF Y=2 THEN *SKIP
1220      LINE(BEFORE(X,0),BEFORE(X,1))-(PX,200-PY),2
1230 '
1240     *SKIP
1250      POINT(PX,200-PY)
1260      BEFORE(X,0)=PX:BEFORE(X,1)=200-PY
1270    NEXT
1280  NEXT
1290 END
1300 '
1310 *FUNCTION
1320  S=(X-42)*(X-42):T=(Y-35)*(Y-35):U=(Y-70)*(Y-70):V=(Y-105)*(Y-105)
1330  F=4500/(S+T+30):G=4500/(S+U+30):H=4500/(S+V+30)
1340  Z=F+G-H
1350 RETURN
```

4章 グラフィック・ツールの紹介

何事でもそうですが，何かよい物を作ろうとすると，それ専用のツールが必要になってきます．そして，そのツールが使用頻度の高いもので，なおかつ使い勝手の良いものであると大幅な省力となり，優れたものを生み出すことができるものです．本章ではそのようなツールをサブルーチン・パッケージの形で紹介します．

4.1 ツールの必要性

上記の例にもれず，コンピュータ・プログラミングにおいてもよく使うルーチンは，1つの汎用サブルーチン・パッケージとして作っておき，ツールとして使うと非常に有益です．1度作っておけば必要となった時に呼び出すだけで済みますし，何より，そのツールの機能さえ分かっていれば，毎回そのプログラムの内容を見る必要はなく，完全なブラックボックスとして使えます．本章では，以降のプログラムの内で使われるもの，また，読者の方々が後で利用価値の大きいものを BASIC と機械語のツールとして提供します．

4.2 BASICによるツール

この節では，N_{88}-BASIC のみで書かれたツールを紹介します．

実際の紹介に入る前に，打ち込みに関するいくつかの注意をしておきます．紹介するツールは，サブルーチン形式になっていて他のプログラムから呼び出して使うものと，何らかの形で他のプログラムにデータを与えるものとなっています．従って，他のプログラムとプログラムなりデータなりをリンクしなければならないのです．Disk-BASIC を使用している場合は，MERGE 命令を使用すればいとも簡単にできるのですが，ROM-BASIC の場合，これに類する機能がありません．ですから，せっかく作ったツールをテープにセーブしておいたとしても，他のテープ上にあるプログラムとの混合が不可能なのです．そこで，打ち込み順序を考えて入力しなければならないことになります．既に完成したツールがテープ上にあるとすれば，まずツールの部分を先にメ

モリにロードしておき，それからメインルーチンを打ち込むといった具合です．この章にも“Gエディタ”など複数のツールを使用するプログラムがありますから，打ち込みの順序を考えてから入力を始めるようにしてください．

この章で紹介するツールは，必ずしもすべてを打ち込む必要はありませんが，前記した様に非常に有用なものですし，当然のことながら，後のプログラムにおいては，指定されているツールがないと実行することができません．なるべく全部打ち込んでください．また，使用にあたっては，常に説明をよく読んで使うようにしてください．

4.2.1 中間色ジェネレータ(MID.SEL)

☆中間色を生成する原理

グラフィックスが視覚的にみて美しいといえる条件として，色の豊富さをその1つに挙げられます．しかし，色の種類を考えますと，わずかに8種類しか用意されていません．ところが，PC-8801は，ドットごとに色を付けることができます．従って，いくつかのドットを1つの点と見なし，その1つ1つのドットに様々な色を与えれば，1点と見なしたドットの集合は，与えられている8色とは多少異なった雰囲気の色を表すはずです（ここでは，このドットの集合である点をピクセルと呼びます）．

しかし，PC-8801程度のドットの細かさでは，ピクセルという考え方で点を打ったり，線を引いたりしても，その効果は期待できません．効果が現れてくるのは，やはり面積を多少なりとも持っている部分を塗ったときでしょう．

では，ピクセルの考え方を用いて，面を塗るにはどうしたらよいでしょう．PSET文を用いて，ピクセルを作成しながら塗りつぶしたとしましょう．膨大な時間を費やすのは明らかですし，境界線の判断も面倒です．そこで，もっと手軽なPAINT文のタイリングによって，このピクセルという概念を実現しながら面を塗ることを考えてみましょう．

気楽に構えていた読者の方は，塗る命令ならPAINT文しかないじゃないかと，考えていたことでしょう．PC-8801のPAINT文は，タイリングという強力な手段を持ち合わせていますので，ピクセルなどと難しい言葉を使わなくても，変わったパターンで面を塗れることはすぐ気付くはずです．ただ，これがある規則に従っているパターンですと，模様というより基本8色以外の色，すなわち中間色に見えるという訳です．ある規則というのが，今まで述べてきたピクセルの概念であることは言うまでもありません．

PAINT文のタイリングにおけるパターンは，タイル・ストリングというストリング変数によって決まります．中間色を用いて色を塗る度に，ピクセルの考えを導入して，タイル・ストリングを作るのでは手間がかかりますし，どんな色が実際の画面上で現れるのか，タイル・ストリングからは想像がつきにくいと思います．どんな中間色が表現できるかを示す何らかの画面上の見本と，それを表現しているデータ（タイル・ストリング）がどんなものであるかを知ることができれば，手間を軽減することができます．そのようなことを一手に引き受けるパッケージが，

リスト1に示される中間色ジュネレータです.

リスト1

```
10000 '>------------------------------------------<
10010 '           << select color package >>
10020 '>------------------------------------------<
10030 '
10040 '  var    DX.M   : cell ﾉ x ﾎｳｺｳ ﾉ ﾄﾞｯﾄ ﾊﾊﾞ
10050 '         DY.M   : cell ﾉ y ﾎｳｺｳ ﾉ ﾄﾞｯﾄ ﾊﾊﾞ
10060 '         SC.M   : display ｦ ﾊｼﾞﾒﾙ ｷｬﾗｸﾀｰ ｻﾞﾋｮｳ
10070 '         MX.M   : cursor ﾉ x ｻﾞﾋｮｳ
10080 '         MY.M   : cursor ﾉ y ｻﾞﾋｮｳ
10090 '         CH.M   : ｷｰﾎﾞｰﾄﾞ ｶﾗ ﾉ ascii code
10100 '         TILE.M$: ﾀｲﾙ ﾊﾟﾀｰﾝ ｦ ｶｴｽ
10110 '         BLUE.M, RED.M, GREEN.M, I.M : ﾜｰｸ ｴﾘｱ
10120 '
10130 '  array TL.M() : ﾀｲﾙ ﾊﾟﾀｰﾝ ﾉ data
10140 '
10150 '>------------------------------------------<
10160 '
10170 ' << middle color initialize >>
10180 '
10190 *MID.INIT
10200  DX.M=16 : DY.M=8 : SC.M=62
10210  OX.M=0 : OY.M=0
10220  RESTORE *TILE.DATA
10230  FOR I.M=0 TO 4
10240    READ TL.M(I.M,0),TL.M(I.M,1)
10250  NEXT
10260  MX.M=SC.M : MY.M=0
10270 RETURN
10280 '
10290 ' << middle color display >>
10300 '
10310 *MID.DISP
10320  FOR GREEN.M=0 TO 4      } 各色バンクごとのループにより
10330    FOR RED.M=0 TO 4      } 各色(三原色)のピクセルに占る割合を変化させる
10340      FOR BLUE.M=0 TO 4   }
10350        X.M=BLUE.M*DX.M+OX.M+SC.M*8 : Y.M=RED.M*40+OY.M
10360        VIEW(X.M,Y.M)-(X.M+DX.M-1.Y.M+DY.M-1).0
10370        GOSUB *MAKE.TILE ←── 対応するタイル・ストリングを生成させる
10380        IF GREEN.M+RED.M+BLUE.M<>0 THEN PAINT(1.1),TILE.M$,7
10390        X.M=X.M+DX.M             ↖黒ならばペイントせずに抜ける
10400      NEXT
10410    NEXT
10420    OX.M=OX.M+DX.M : OY.M=OY.M+8
10430  NEXT
10440  SCREEN 0,0
10450  BLUE.M=0 : RED.M=0 : GREEN.M=0
10460 RETURN
10470 '
10480 ' << tile pattern making >>
10490 '
10500 *MAKE.TILE                       ↙タイル・ストリングの生成
10510  TILE.M$=CHR$(TL.M(BLUE.M,0))+CHR$(TL.M(RED.M,0))+CHR$(TL.M(GREE
N.M,0))+CHR$(TL.M(BLUE.M,1))+CHR$(TL.M(RED.M,1))+CHR$(TL.M(GREEN.M,1))
10520 RETURN
10530 '
10540 ' << cursor control >>
10550 '
10560 *MID.SEL
10570  CONSOLE 0,25 : LOCATE MX.M,MY.M
10580  CH.M=ASC(INPUT$(1))
10590 '
10600 *LP.M : IF CH.M=&H1B THEN *EX.M
10610     IF CH.M=29 THEN BLUE.M=BLUE.M-1:IF BLUE.M<0 THEN BLUE.M=4
10620     IF CH.M=28 THEN BLUE.M=BLUE.M+1:IF BLUE.M>4 THEN BLUE.M=0
10630     IF CH.M=30 THEN RED.M=RED.M-1:IF RED.M<0 THEN RED.M=4
10640     IF CH.M=31 THEN RED.M=RED.M+1:IF RED.M>4 THEN RED.M=0
10650     IF CH.M=13 THEN GREEN.M=GREEN.M+1:IF GREEN.M>4 THEN GREEN.M=0
10660     MX.M=SC.M+(BLUE.M+GREEN.M)*DX.M¥8 : MY.M=RED.M*5+GREEN.M
10670     LOCATE MX.M,MY.M
10680     CH.M=ASC(INPUT$(1))
```

```
10690 GOTO *LP.M
10700 '
10710 *EX.M : GOSUB *MAKE.TILE
10720 RETURN
10730 '
10740 ' << source tile data >>
10750 '
10760 *TILE.DATA
10770  DATA 0,0,136,34,170,85,238,187,255,255
```

☆プログラムの説明と使い方

このパッケージは，中間色の見本の表示，その選択，さらに選択された中間色のタイル・ストリングの作成を行います．ではサブルーチンごとにその機能を説明しながら，使用法などにも触れていきましょう．

＊MID．INIT は，このパッケージのイニシャライズ（初期化）をする所です．従って，このサブルーチンは，パッケージの中の他のサブルーチンを使用する前に必ず呼び出しておいてください．言わばパッケージの使用開始宣言とでもいったところです．ここで初期化される変数や，その他パッケージの内部で用いられている変数，配列等は，パッケージの外部でその内容を破壊しないように注意してください．パッケージの内部で用いた変数などには，分かり易くするため，末尾に〝．M″の拡張子を付けてあります．ユーザー（読者の皆さん）が変更できる変数は，DX．M と SC．M の2つだけです．

DX．M は，中間色の見本の1つの横幅のドット数です．これは8の倍数にしておく必要があります．縦幅は，DY．M が示し，8ドットに固定です．SC．M は，中間色の見本を描き始める場所を指定するもので，キャラクタ座標のx軸方向（横軸方向）の値です．但し，WIDTH 80, 25の状態で，パッケージが実行されることを前提としています．このDX．M と SC．M の値と画面上の表示との関係は＊MID．DISP のところで詳しく説明します．

変数への代入文が2行続いたあとで，TL．M という配列に，READ 文でデータを拾っています．実はこのデータこそが，中間色を生成するときに必要だったピクセルの考え方を実現させるパターンなのです．そのパターンを明らかにするために，＊TILE．DATA の後に続くデータを解析してみましょう．

このデータは10進数で示してありますが，ドット・パターンを考え易くするために，16進数に変換してみましょう．表1にその様子を示します．

このパッケージでは，1つのピクセルが図1に示すような並び方の4つのドットで成り立っています．表1の縦の並びの組み合わせを，いくつか並べてみてください．例えば，238と187，つまり16進数で EE とBB とを交互に縦に並べてみると，図2-(a)のようになります．このパターンから，図1に示したようなドットの並びの部分に着目してみましょう．すると，図1の形にとった部分は，そのどれもが図2-(b)のどれかになります．

すなわち，1つのピクセル（この場合，4つのドットから成る）の4分の3のドットを必ず選択していることになります．ですから，これが青のプレーンにおいて作用すれば，完全な青（4ドットとも青）の75％の明るさの青を示すはずです．他のデータも同様に考えると，0％，25％，

TL.M(a,n)

n ＼ a		0	1	2	3	4
0	10 進 数	0	136	170	238	255
	16 進 数	0	88	AA	EE	FF
	ドット・パターン	□□□□□□□□	■□□□■□□□	■□■□■□■□	■■■□■■■□	■■■■■■■■
1	10 進 数	0	34	85	187	255
	16 進 数	0	22	55	BB	FF
	ドット・パターン	□□□□□□□□	□□■□□□■□	□■□■□■□■	■□■■■□■■	■■■■■■■■

■が点灯

表 1

ピクセルの考え方．4つのドットを1つの点と見なす．

図 1

75%選択されたピクセルの組み合わせ

図2(a)

75%選択されたピクセルのパターン

図2(b)

サブルーチン＊MID・DISPによる画面の表示

図 3

50%，75%，100%という5種類のピクセルが，各プレーンについて表現できる訳ですから，PC-8801の3つのプレーンの場合ですと，5^3＝125種類の色を表現できることになります（ここでは，1つのピクセルを4ドットとしましたが，実際には，2でも9でも構いません．但し，発色する数や美しさから考えて4が最もベストの様です）．

さて，これら125色の中間色を，＊MID．DISPを呼び出して画面上に表示します．表示し始める位置は，＊MID．INITの所で説明したSC．Mによって決定されています．しかし，SC．Mが示す値というのはキャラクタ座標ですから，これをグラフィック座標に直すために，8倍する必要があります．それを行っているのが，1360行のX．Mへの代入文です．

FORループを回っている時に，BLUE．M，RED．M，GREEN．Mが，それぞれ保持している値の0～4は，先程の0～100%の5つの値に対応しています．このことは表1を見ても明らかです．BLUE．M＝1，RED．M＝2，GREEN．M＝3だとすれば，青のプレーンが25%，赤のプレーンが50%，緑のプレーンが75%，選択された色合いの中間色ができる訳です．その中間色を示すタイル・ストリングを作成するのが＊MAKE．TILEで，1380行で呼び出しています．

図3を見てください．＊MID．DISPは，このように中間色を並べてくれます．1段表示する度に，右側の方にDX．Mの示すドット数ずつずれて表示していくので，そのことも考えて画面をはみ出さないように，SC．Mの値を決める必要があることを忘れないでください．リスト1のまま使用するのが比較的使い易いようです．図3の矢印の方向は，図に描かれている色のプレー

ンの選択される割合の強い方向です．但し，赤は1つの平行四辺形ごとに，その割合の強さが変化するようになっています．このことは，中間色を実際選んでみれば理解して頂けるでしょう．

その中間色を選択する処理は，＊MID．SEL 以降のルーチンで行います．このルーチンを使用するのに重要なことは，まず，1580行に示されているように，今まで CONSOLE 文で指定されていたものを破壊しますので，このルーチンから戻った時は設定し直さなければならないことです．＊MID．SEL を呼び出すと，表示した中間色の左上の部分にカーソルが出ます．そこで，[→]を押すとカーソルは右に動きます．ではさらに4回[→]を押してみてください．カーソルはもとに戻ったと思います．もうお分かりのように，[→]キーは，青の量を変化させる方向にカーソルを動かします(しかも，カーソルの動きはループ状です)．[←]キーは，その逆ですから試してみてください．

次に，赤の量を変化させてみましょう．気の早い方は，[↓]キーや，[↑]キーを押しているかもしれませんが，その通りです．[↓]キーや[↓]キーは，赤の量を変化させる方向にカーソルを動かすことができるキーで，これもループ状になっています．このキーによるカーソルの動きは，各平行四辺形のブロックごとの移動になっていますので，画面の下の平行四辺形のブロック程，赤の割合が強いということです．言い換えれば，平行四辺形の各ブロックの中では，赤の割合は同じということです．では，緑の割合はどうやって変化させるのでしょう．

それはリターン・キー（[⏎]）です．リターン・キーを押すと，緑の割合が増す方向にカーソルが移動し，平行四辺形の中でループ状に動きます．これには，残念ながら逆方向に動かす機能がありません．

さあいかがでしょうか．もう思い通りの中間色の所に，カーソルを移動できるようになったでしょうか．キーとカーソルの動きを，表2にまとめてみました．

中間色を選び終えたら，[ESC]キーを押してください．文字通り，＊MID．SEL のルーチンから抜け出してきます．このルーチンから戻る際に，TILE．M$ の中に選んだ中間色のタイル・ストリングを持ってきますので，これを用いて任意の部分を塗れる訳です．では使用例を挙げてみましょう．リスト1の最初に，次の7行（リスト2）を加えて，実行してみてください．

使用法――お分り頂けたでしょうか．決して難しいものではないと思いますし，中間色の選択の仕方も，慣れれば結構便利でしょう．有効に利用して頂けたら幸いです．

[→] [←]	各段の中で，左右にカーソルを動かします．
[↓] [↑]	平行四辺形のブロックを選択します．
[⏎]	各平行四辺形のブロック内のどの段かを選択します．

表2　パッケージ，＊MID.SELにおける色の選択法

リスト2

```
10  ***selcol sample ***
20  SCREEN 0,0 : CLS 3
30  WIDTH 80,25 : CONSOLE 0,25,0,1
40  GOSUB *MID.INIT
50  GOSUB *MID.DISP
60  CIRCLE(200,100),50,7
70  GOSUB *MID.SEL
80  PAINT(200,100),TILE.M$,7
90 END
```

4.2.2 タイル・パターン・ジェネレータ

4.2.1では，PAINT 命令のタイリング機能を使って疑似中間色を作りましたが，これはタイリング機能のある特別な（ある一定の規則にのっとった）場合でした．ここでは，本来のタイリングの機能に注目し，様々な模様やパターンを作ることができるツール（サブルーチン）を紹介します．タイル・ストリングの基本的な作り方は，〝中間色ジェネレータ〟のところで説明しましたから，ここでは詳しく触れません．

☆タイル・パターンの作り方

まずタイル・パターンを作るには，いくつかの方法があると思います．最も手取り早い方法は，色鉛筆と方眼紙を持ってきて実際に紙の上でパターンを作り，各ラインのデータを目で読むことでしょう．しかし，これではあまりにも芸がありません．グラフィック画面に PSET 命令で点をセットしながら作ることも考えられます．この方法だと画面上にデータが残りますから，タイル・ストリングの生成はコンピュータに任せることができるでしょう．しかし，実際にやってみると分かりますが，PC-8801の画面の分解能が，あまりにも細かいため，使い物にならないでしょう．そこで今回は，PC-8801のグラフィック面画とキャラクタ画面が個別であることを利用して，キャラクタ面画でタイル・パターンを作ることにしました．

原理としては，キャラクタ画面上でカーソル・キー（移動用）と数字キー（カラー指定用）を使いタイル・パターンをカラーで作っていきます．その際描かれたパターンをその都度2次元配列に保存しておき，パターンを描き終った時点で配列の内容を参照し，画面のパターンに対応したタイル・ストリングを生成するのです．

☆プログラムの説明

では，この事を頭に入れてプログラムを見てください．このプログラムの各変数及びラベル名のほとんどは，他のメイン・ルーチンから呼ばれることを考慮して〝.T〟の拡張子が付けてあります．まず，最初の＊TILE．INIT は，このプログラムの初期設定の部分です．ここでキャラクタの初期座標や配列の大きさなどを設定していきます．このルーチンは，このプログラムを使う前に1回だけ呼び出しておけば結構です．

リスト3

```
20000 '
20010 '     >> tile pattern generator <<
20020 '
20030 *TILE.INIT
20040  CX.T=65 : CY.T=0              ┐
20050  MODE.T=3 : CHAR.T$="*"        ├ 初期設定
20060  DIM TILE.T(7,21)              ┘
20070 RETURN
20080 '
20090 '   << frame draw & define max y >>
20100 '
20110 *TILEGEN
20120  CLS : COLOR 7 : LOCATE ,,1 : CONSOLE 0,25
20130  INPUT "max y";YMAX.T
20140  IF YMAX.T<1 OR YMAX.T>22-CY.T THEN *TILEGEN
20150  BLACK.T=0 : XMAX.T=7
20160  LOCATE CX.T,CY.T
20170  PRINT "┌────────┐"
20180  FOR I.T=1 TO YMAX.T
20190    LOCATE CX.T,CY.T+I.T
20200    PRINT "│        │"
20210  NEXT
20220  LOCATE CX.T,CY.T+YMAX.T+1
20230  PRINT "└────────┘"
20240  YMAX.T=YMAX.T-1
20250  GOSUB *TILE.PAT
20260 RETURN
20270 '
20280 '  << make tile pattern >>
20290 '
20300 *TILE.PAT
20310  SCREEN ,2 ←グラフィック画面をOFF
20320  LOCATE CX.T+1,CY.T+1
20330  X.T=0 : Y.T=0
20340  CH.T=ASC(INPUT$(1))
20350 '
20360 *LOOP0.T : IF CH.T=&H1B THEN *EXIT0.T
20370     IF CH.T=29 THEN X.T=X.T-1:IF X.T<0 THEN X.T=XMAX.T
20380     IF CH.T=28 THEN X.T=X.T+1:IF X.T>XMAX.T THEN X.T=0
20390     IF CH.T=30 THEN Y.T=Y.T-1:IF Y.T<0 THEN Y.T=YMAX.T
20400     IF CH.T=31 THEN Y.T=Y.T+1:IF Y.T>YMAX.T THEN Y.T=0
20410     IF (CH.T<&H30) OR (CH.T>&H37) THEN *EXIT.T
20420     C.T=CH.T-&H30
20430     IF C.T=0 THEN COLOR 7 : PRINT " " ELSE COLOR C.T : PRINT CHAR
.T$
20440     TILE.T(X.T,Y.T)=CH.T-48
20450   '
20460   *EXIT.T
20470    LOCATE CX.T+X.T+1,CY.T+Y.T+1
20480    CH.T=ASC(INPUT$(1))
20490  GOTO *LOOP0.T
20500 *EXIT0.T : GOSUB *TILE.STR
20510  SCREEN ,0
20520 RETURN
20530 '
20540 '  << make tile string >>
20550 '
20560 *TILE.STR
20570  COLOR 7 : LOCATE CX.T,CY.T
20580  TILE.T$="" : Y.T=0
20590 '                                       y方向(行数)のループ
20600 *LOOP1.T : IF Y.T>YMAX.T THEN *EXIT1.T ↙
20610    X.T=0 : BLUE.T=0 : RED.T=0 : GREEN.T=0
20620   *LOOP2.T : IF X.T>=8 THEN *EXIT2.T ←x方向(桁数)のループ
20630      BLUE.T=BLUE.T*2+(TILE.T(X.T,Y.T) AND 1)         ┐ 1，2，4でANDを取る
20640      RED.T=RED.T*2+(TILE.T(X.T,Y.T) AND 2)¥2         ├ ことにより，下位3bitの
20650      GREEN.T=GREEN.T*2+(TILE.T(X.T,Y.T) AND 4)¥4     ┘ ON／OFFを調べる
20660      X.T=X.T+1
20670    GOTO *LOOP2.T
20680   *EXIT2.T : TILE.T$=TILE.T$+CHR$(BLUE.T)
20690    IF MODE.T=1 THEN *NEXT.T ←── もし白黒パターンの場合は，BLUT.Tしか考慮しない．
20700    TILE.T$=TILE.T$+CHR$(RED.T)+CHR$(GREEN.T) 結果としてタイル・ストリング
20710   *NEXT.T : Y.T=Y.T+1                         をTILE.T$に入れて持ち返る
```

```
20720  GOTO *LOOP1.T
20730 *EXIT1.T : RETURN
20740 '
20750 ' << display tile string data >>
20760 '
20770 *DATA.DISP ←―― タイル・ストリングの内容をキャラクタ・コードに変換し，
20780  CLS : C.T=1                          DATA文として表示する
20790  FOR I.T=0 TO YMAX.T
20800    PRINT "data ";
20810    FOR J.T=1 TO MODE.T
20820      IF J.T<>1 THEN PRINT ",";
20830      PRINT RIGHT$("0"+HEX$(ASC(MID$(TILE.T$,C.T,1))),2);
20840      C.T=C.T+1
20850    NEXT
20860    PRINT
20870  NEXT
20880 RETURN
```

＊TILEGEN 以降 RETURN までが，このプログラムのメイン・ルーチンです．まず，作るタイル・パターンのサイズを決定します．横方向は PAINT 命令の制約上 8 ですが，縦方向は可変ですから INPUT 文でユーザーに決定を求めています．但し，キャラクタ画面の制約上22を最大としています．後はサイズの大きさを示す枠を描き，実際にパターンを描くルーチンに制御を移します．

＊LOOP0．T から EXIT0．T までは，カーソルを動かすルーチンとキャラクタを表示するルーチンです．パターンを描き終り，このルーチンを抜けると，配列 TILE．T(　)に確保されている各ドットのための色情報を解析し，目的のタイル・ストリングを得ています．この時，配列内の 1 要素がタイル・ストリングの 3 ビット（白黒モードでは 1 ビット）に，配列内の横方向の 1 ラインがタイル・ストリングの 3 キャラクタ（白黒モードでは 1 キャラクタ）に圧縮されます．1630行〜1650行では配列内のカラー・コードの値により，三原色の ON/OFF を調べています．この過程を図 4 に示します．結果的に，この 2 つのループ内で得られたタイル・キャラクタを頭からつなぎ合わせてタイル・ストリングとし，メインルーチンに持ち帰ります．

※ 7より大きい値はないため
bit3〜bit6はすべて0

	赤	赤	白	白	緑	黄	青	青	
	2	2	7	7	4	6	1	1	カラー・コード
bit 2	0	0	1	1	1	1	0	0	緑のデーター→ &H3C
bit 1	1	1	1	1	0	1	0	0	赤のデーター→ &HF4
bit 0	0	0	1	1	0	0	1	1	青のデーター→ &H33

（白黒モードの場合bit0のみ考慮する）

↑最上位　各ライン(8個)を1バイトデータと見なす　↑最下位

図4　カラー・モードにおけるMAX．Y＝1の場合のタイル・ストリングの生成

このプログラムでは，＊DATA．DISP のルーチンは呼んでいませんが，このルーチンは求めたタイル・ストリングの内容をキャラクタ・コードとして16進数で表示するものです．作ったタイル・パターンのストリングを書き留めておき，他のプログラムで使います．また，表示されたデータにスクリーン・エディタで行番号を付けプログラムの１部とするのもよいでしょう．このルーチンが必要となった時は，1260行の RETURN の前で GOSUB　＊DATA．DISP を実行するか，メイン・プログラムから呼び出してください．

現在，このプログラムはカラー・モード用ですが，＊TILE．INIT 内の MODE．T を１にすることにより，白黒モード用としても使用可能です．1680行～1700行のタイル・ストリングの生成において，代入文を２つに分けているのはそのためです（白黒時にはBLUEの部分——最下位ビットのみ——しか考慮しない）．

☆プログラムの呼び出し方

基本的には，

```
GOSUB  *TILE.INIT
GOSUB  *TILEGEN
```

で OK ですが，キャラクタ画面がカラー・モードであり，なおかつ80×25行モードであることを前提としています．引き渡すパラメータはなく，結果として生成したタイル・ストリングを TILE．T$ として持ち返ります．このプログラムでは，グラフィック画面を壊さないように考慮してありますが，キャラクタ画面は CLS を行い，画面の右端をワーク画面としています．もし，このワークの位置を変更したい場合は，＊TILE．INT 内の CX．T を書き換えることにより可能です．では，呼び出し方のサンプルとして，１例をリスト４に示しておきます．

リスト4

```
100 '
110 '   sample program for tile pattern generator
120 '
130  SCREEN 0,0 : CLS 3
140  WIDTH 80,25 : CONSOLE 0,25,0,1
150  CIRCLE(320,100),100,7
160  GOSUB *TILE.INIT
170  GOSUB *TILEGEN
180  GOSUB *DATA.DISP
190  PAINT (320,100),TILE.T$,7
200 END
```

☆**プログラムの使い方**

使い方は簡単で，最初の〝max　y″の問に答えた後，枠の内をカーソル･キーで移動しながら，カラー・コードに対応する数字キーでキャラクタ〝*″を配置し，パターンを作っていきます．もし，間違った箇所を修正する場合は，〝0″のキー（黒）で消します．パターンを作り終わったら〝ESC″キーで抜けてください．白黒用のパターンを作成する場合は，〝7″のキー（白）で行ってください．

4.2.3　シンボル・ジェネレータ

シンボル･ジェネレータ（リスト5）は，グラフィック画面上にPUTする比較的小さなパターン用の配列データを作成するプログラムです(サブルーチンではありません)．まず，前項のタイル・パターン・ジェネレータと同じくキャラクタ画面で，キャラクタによりパターンを作成し，そのキャラクタ1文字を1ドットと見なしてパターンを作成します．そして，そのパターンをグラフィック画面に転送した後，GET し，そのときの配列要素を出力します．

出力は，次の3種類の形態で行えます．

① 配列要素の内容のデータ・ファイル
② 画面上の DATA 文
③ マージ用のプログラム・ファイル

①は，配列の内容をそのままシーケンシャル・ファイルに落すもので，シンボル・パターンを必要とするプログラムで，そのファイルを読み込まなければなりません．これはディスク／カセット共に利用可能です．

②は画面上に必要なデータを収めた行番号付きの DATA 文を表示し，プログラムの実行終了後，カーソルを DATA 文の先頭に持っていき，リターン・キーを押して，プログラムの1部としてしまう方法です．後は，ジェネレータの部分を DELETE して，プログラムの他の部分を入力していきます．

③は Disk-BASIC の場合のみ使用可能な方法で，他のプログラムにマージして使うための，アスキーセーブされた DATA 文のみのプログラム・ファイルをディスク上に作成します．

これらのデータは，すべて整数配列で扱っているため，読み込むときは整数型で行ってください．

リスト5

```
30000 '
30010 '   SYMBOL DATA GENERATOR
30020 '
30030  DEFINT A-Z
30040 *SIZE.LOOP.S
30050  INPUT "SIZE (X,Y)";X.S,Y.S
30060  IF (X.S>80) OR (Y.S>24) THEN *SIZE.LOOP.S
30070  X1.S=X.S-1 : Y1.S=Y.S-1
30080  CT.S=(((X.S+7)¥8)*Y.S*3+4)¥2+1
30090  DIM CLR.S(X1.S,Y1.S),PAT.S(CT.S)
```

```
30100 *MODE.LOOP.S
30110  INPUT "FILE or DATA or PROGRAM";MODE.S$
30120  MODE.S$=LEFT$(MODE.S$,1)
30130  IF (MODE.S$<>"D") AND (MODE.S$<>"F") AND (MODE.S$<>"P") THEN *M
ODE.LOOP.S
30140  IF (MODE.S$="D") OR (MODE.S$="P") THEN INPUT "LINE NUMBER";LN.S
!
30150  IF (MODE.S$="F") OR (MODE.S$="P") THEN INPUT "FILE NAME";FILE.N
AM.S$
30160  SCREEN 0,0
30170  CLS 3
30180  LINE(0,0)-(8*X.S,8*Y.S),7,B
30190  CX.S=0 : CY.S=0
30200 *INP.LOOP.S
30210    A.S=ASC(INPUT$(1))
30220    IF (A.S>=&H30) AND (A.S<=&H37) THEN *DOT.SET.S
30230    IF A.S=&H1D THEN *LFT.S
30240    IF A.S=&H1C THEN *RGT.S
30250    IF A.S=&H1E THEN *UP.S
30260    IF A.S=&H1F THEN *DWN.S
30270    IF A.S=&H1B THEN *GEN.PAT.S
30280  GOTO *INP.LOOP.S
30290 *LFT.S
30300    CX.S=CX.S-1
30310    IF CX.S=-1 THEN CX.S=X1.S : GOTO *UP.S
30320    LOCATE CX.S,CY.S
30330  GOTO *INP.LOOP.S
30340 *RGT.S
30350    CX.S=CX.S+1
30360    IF CX.S=X.S THEN CX.S=0 : GOTO *DWN.S
30370    LOCATE CX.S,CY.S
30380  GOTO *INP.LOOP.S
30390 *UP.S
30400    CY.S=CY.S-1
30410    IF CY.S=-1 THEN CY.S=Y1.S
30420    LOCATE CX.S,CY.S
30430  GOTO *INP.LOOP.S
30440 *DWN.S
30450    CY.S=CY.S+1
30460    IF CY.S=Y.S THEN CY.S=0
30470    LOCATE CX.S,CY.S
30480  GOTO *INP.LOOP.S
30490 *DOT.SET.S
30500    CL.S=A.S-&H30                                        画面への表示
30510    IF CL.S=0 THEN PRINT " "; ELSE COLOR CL.S:PRINT "*";←
30520    COLOR 7
30530    CLR.S(CX.S,CY.S)=CL.S ←── パターン記憶用の配列へカラー・コードを代入
30540  GOTO *RGT.S
30550 *GEN.PAT.S
30560    CLS 3
30570    FOR I.S=0 TO X1.S
30580      FOR J.S=0 TO Y1.S
30590        PSET (I.S,J.S),CLR.S(I.S,J.S)  } グラフィック画面へパターンを表示
30600      NEXT
30610    NEXT
30620    GET (0,0)-STEP(X1.S,Y1.S),PAT.S
30630    SCREEN 0,0
30640    COLOR 7
30650    IF MODE.S$="F" THEN *FILE.S
30660    IF MODE.S$="P" THEN OPEN FILE.NAM.S$ FOR OUTPUT AS #1
30670    K.S=0
30680   *GEN.LOOP.S
30690      IF (K.S MOD 8)=0 THEN ST.S$=STR$(LN.S!)+" DATA"
30700      ST.S$=ST.S$+STR$(PAT.S(K.S))
30710      IF (K.S MOD 8)<>7 THEN ST.S$=ST.S$+"," ELSE GOSUB *OUTPUT.S
 : LN.S!=LN.S!+10
30720      K.S=K.S+1                                        データ文作成
30730      IF K.S<>CT.S THEN *GEN.LOOP.S
30740      IF RIGHT$(ST.S$,1)="," THEN ST.S$=LEFT$(ST.S$,LEN(ST.S$)-1)
30750      IF LEN(ST.S$)<>0 THEN GOSUB *OUTPUT.S
30760 END
30770 '
30780 *FILE.S
```

```
30790     OPEN FILE.NAM.S$ FOR OUTPUT AS #1 ┐
30800     FOR I.S=0 TO CT.S                 ├ シーケンシャル・ファイル作成
30810       PRINT #1,PAT.S(I.S)             │
30820     NEXT                              ┘
30830 END
30840 '
30850 *OUTPUT.S
30860  IF MODE.S$="P" THEN PRINT #1,ST.S$ ←── ファイルに出力
30870  IF MODE.S$="D" THEN PRINT ST.S$    ←── 画面に出力
30880  ST.S$=""
30890 RETURN
```

```
10000 DATA 35, 10, 0, 0, 0, 0, 0, 0
10010 DATA 0, 0, 0, 0, 0, 0, 0, 0
10020 DATA 0, 0, 0, 0,-1793, 0, 0, 0
10030 DATA 0,-1795, 16, 515, 4, 0,-1795, 16
10040 DATA-765, 2300, 3072, 514, 0,-765, 2300, 0
10050 DATA 0, 0,-225,-1, 0, 0, 0, 64
10060 DATA 0, 32576,-1,-16129, 64, 0, 16448, 0
10070 DATA 8192,-129,-1, 16600, 0, 8192, 0, 0
10080 DATA 7936,-1, 255,-16377, 31744, 0, 0, 0
10090 DATA-32765, 14336, 768, 128, 56, 0
Ok
```

では，このプログラムの使用法を説明しましょう．このプログラムで使う変数名，ラベル名には，〝.S〞という拡張子がついています．

☆プログラムの使い方

起動すると，作成するシンボルのサイズを聞いてきます．横は最大80，縦は最大24です．次に3つの出力形態のどれを選ぶか聞いてきます．〝FILE〞がデータ・ファイルの作成，〝DATA〞が画面に DATA 文を表示，PROGRAM がマージ用のプログラム・ファイルの作成です．〝F〞か〝D〞か〝P〞を大文字で入力してください．それが済むと，必要に応じて，ファイル・ネームや開始行番号を聞いてきますから，入力してください．

これらが終わると，画面に入力したサイズに応じた四角形が表示されます．カーソル・キーにより，カーソルがこの中を移動し，数字の0から7で，その値をカラー・コードとする〝*〞がカーソルの位置に表示されます．このとき，プログラムでは，画面に〝*〞を表示すると共に表示画面と同じ大きさの配列にカラー・コードを書き込み，表示画面のコピーを作っています．この操作は，タイル・ストリング・ジェネレータとほぼ同じです．

そして，この配列の内容をカラー・コードとして，グラフィック画面上に PSET 文でシンボルを表示し，それを配列に GET し，データを作成します．

このモードからの抜け出しは〝ESC〞キーで行い，最後にデータを出力します．

このプログラムを使う上での注意を2つあげておきます．横方向を長く取り，途中で色を変えすぎると，アトリビュートの都合で，押したキーの値と表示される〝*〞の色が合わなくなることがあります．これは，ハードウェアの構造上やむを得ないことなのですが，作成されるデータはキーの値に合ったものになります．

DATA 文の画面表示を指定したとき，画面サイズが大きすぎると，DATA 文の行数が多くなり，スクロール・アップして，最初の方の DATA 文が失われてしまいます．このようなときは，別の出力モードに切り換えるか，シンボルを分割して作成するようにしてください．

リスト6に，このプログラムで作成したシーケンシャル・ファイルを読み込むサブルーチンを示しておきます．

リスト6

```
40000 *INIT.SG
40010 '
40020 '  SET PATTERN DATA
40030 '
40040  T.SG=XN.SG-1
40050  DIM X.SG(T.SG),SYM.SG%(147),TITLE.SG%(24)
40060 *INIT2.SG
40070  OPEN FILE.NAM.SG$ FOR INPUT AS #1
40080  FOR I.SG=0 TO 147
40090    INPUT #1,SYM.SG%(I.SG)
40100  NEXT
40110  CLOSE #1
40120 RETURN
```

4.2.4 グラフィック・エディタ

☆グラフィック・エディタの概略

N_{88}-BASIC の持つ数々のグラフィック命令や，今まで説明してきたツール(中間色ジェネレータ，タイル・パターン・ジェネレータ)，さらには，4．3で述べる機械語によるグラフィックのツールなどは，単体で存在していても面白味に欠ける所があります（ツールだけでは，ほとんど意味を持たないと言っても過言ではありません)．画面上に絵を描くにしても，N_{88}-BASIC のグラフィック命令だけでは，方眼紙に絵を描いて，その座標をデータとして拾うといった程度の味気ない作業になってしまいます．ペンを持って紙に描く様にとはいかないまでも，画面上の任意の所に点や線が描ければ，方眼紙上のデータを拾うよりもかなり楽で，しかも便利であることは間違いありません．そこで，画面に絵や様々なパターンを描くために用意したツールが，グラフィック・エディタ（以後Gエディタと呼ぶ）です．実際このプログラムは，単独で機能するものですが，色々なパターンを作って他のプログラムで利用可能ですから，あえてツールと呼んでいます．冒頭のカラーページの〝桃太郎〞の絵は，このGエディタで作成したものです．

以前に説明したツールと，N_{88}-BASIC のグラフィック命令のすべてを網羅していますので，プログラムによって表現できる図形――時としてその以上の絵や図形を表すことができるはずです．描く方法は，機械語のパッケージ(4．3で紹介)によってサポートされているクロス・カーソル（画面上に縦と横の青い線が交わっているもの．図5を参照）を用いて，そのクロス・カーソルの交わっている点（着目している点）について，種々の命令を与えることによって行います．命令は，すべてキーボードから入力していく形をとっており，特別な入力装置を必要としません．その他は対話形式で処理していきます．

プログラムをリスト7に示します．かなり長いプログラムですので，ポイントとなる所だけピックアップして，詳しく説明することにしましょう．

表示域

点線内で自由に動く

図5　クロス・カーソル

但し，このプログラムでは，〝中間色ジェネレータ〟及び〝タイル・パターン・ジェネレータ〟及び4.3で紹介する機械語パッケージを必ず必要とします．

リスト7

```
1000 '
1010 ' << Graphic Editor >> 注意）必ずGlib,MID.SEL,TILEGENが必要
1020 '
1030  CLEAR ,&HE3FF              ┐
1040  GOSUB *INITIALIZE          │メイン・ルーチン
1050  GOSUB *MAIN.LOOP           │
1060 END                         ┘
1070 '
1080 *INITIALIZE
1090  WIDTH 80,25 : CONSOLE 20,5,0,1
1100 '
1110  DEFINT A-Z
1120  DEF USR=&HE400
1130 '
1140  ON ERROR GOTO *TRAP
1150  ON HELP GOSUB *MSG.BLOCK : HELP ON
1160 '
1170 *N88.SYSTEM
1180  COLOR 7 : SCREEN ,2 : CONSOLE 0,25 : CLS                        ┐使用している
1190  LOCATE 25,7 : PRINT "Your N88-BASIC system is ....."             │N88-Basicが
1200  LOCATE 25,9 : PRINT "     1) Disk version"                       │Disk BasicかRom
1210  LOCATE 25,10 : PRINT "     2) Rom version"                       │Basicか?
1220  LOCATE 25,12 : INPUT "   number please.  ",BASIC                 │
1230  IF BASIC<>1 AND BASIC<>2 THEN *N88.SYSTEM                        ┘
1240 '
1250  GOSUB *GR.INIT
1260  GOSUB *PAINT.INIT
1270 RETURN
1280 '
1290 *MAIN.LOOP
1300  SCREEN 0,0 : CONSOLE 20,5,0,1
1310  WINDOW(0,0)-(XW.G,YW.G) : VIEW(0,0)-(XW.G,YW.G)
1320  GOSUB *LPSET.G
1330  GOSUB *INFORM.G
1340 '
1350  DUMMY=USR(GCREV.)クロス・カーソルの表示
1360  GOSUB *IN.KEY        キー・コマンドの入力
1370 *LP.G : IF CH$=CHR$(&H1B) THEN *EX.G  [ESC] キーなら*EX.Gへ
1380    IF CH$="" THEN *CURSOR.G                      ┐
1390    VEC.=INSTR(" srlbfcpeamGPSL><",CH$)           │コマンドの解析
1400    IF VEC.=0 THEN *CURSOR.G                      ┘
1410 '                                                              各処理への振り分け
1420    DUMMY=USR(GCREV.)クロス・カーソルの消去
1430    ON VEC. GOSUB *COLOR.G,*PSET.G,*PRESET.G,*LINE.G,*LINEB.G,*LIN ┐
EBF.G,*CIRCLE.G,*PAINT.G,*CLS.G,*LPSET.G,*MODE.G,*GET.AT.G,*PUT.AT.G,* │
SAVE.G,*LOAD.G,*REVERSE.G,*RETURN.G                                    ┘
1440    DUMMY=USR(GCREV.) クロス・カーソルの表示
1450 '
1460   *CURSOR.G                                    モードによるクロス・カーソル
1470    IF MODE.G=1 THEN PRESET(X,Y)                の制御                    ┐
1480    IF MODE.G=2 THEN PSET(X,Y)                                            │
1490    IF MODE.G=3 THEN DUMMY=USR(GCREV.) : R!=R!+.4:CIRCLE(X,Y),R! :       ┘
 DUMMY=USR(GCREV.)
```

```
1500     DUMMY=USR(GCMOV.) クロス・カーソルの移動
1510     IF DUMMY=255 THEN R!=0
1520    'LOCATE 0,20 : PRINT USING "[ x=###  y=### ]";X,Y
1530 '
1540     GOSUB *IN.KEY キー・コマンドの入力
1550 GOTO *LP.G
1560 *EX.G : DUMMY=USR(GCREV.) クロス・カーソルの消去
1570 '
1580  CLS : INPUT "Copy to printer ";A.G$
1590  IF A.G$<>"y" THEN *MORE
1600  PRINT "Prepare your printer !"
1610  PRINT "Press any key."
1620 *WAIT.LOOP : IF INKEY$="" THEN *WAIT.LOOP
1630 'COPY 2      ハード・コピー
1640 '
1650 *MORE
1660  PRINT : PRINT "Retry ?";
1670  GOSUB *GET.KEY : PRINT
1680  IF CH.G=ASC("y") THEN *MAIN.LOOP
1690  ON ERROR GOTO 0      エラー・トラップの解除
1700  COLOR 7 : CLS : PRINT "See you again !"
1710  CONSOLE 0,25
1720 RETURN
1730 '
1740 *GET.KEY                     ┐
1750  CH.G=ASC(INPUT$(1))         │
1760 RETURN                       │
1770 '                            ├キーの入力に関する操作
1780 *IN.KEY                      │
1790  CH$=INKEY$                  │
1800 RETURN                       ┘
1810 '
1820 '  << ｸﾞﾗﾌｨｯｸ ﾒｲﾚｲ ﾉ ｲﾆｼｬﾗｲｽﾞ  >>
1830 '
1840 *GR.INIT
1850  CLS : COLOR 1,0,0,7
1860  XW.G=485 : YW.G=154 : C.G=7
1870 'X=XW.G/2 : Y=YW.G/2 ←――― 着目座標のイニシャライズ
1880 '
1890  VEC.=0 : MODE.G=0 : R!=0 : CH.G=0 : R.G=0
1900  P.G=0 : B.G=0 : YMAX.T=0 : XMAX.T=0 : I.T=0 : C.T=0 }使用する変数の登録
1910  CH.M=0 : CH.T=0 : CH$="" : FILE.G$="" : MSG$=""
1920 'FL.G$="" : TILE.M$="" : TILE.T$=""
1930 '                                                    グラフィック・ライ
1940  CLSM.=0 : CLSS.=1 : CLSH.=2 : GCREV.=14 : GCMOV.=15 ブラリに関するイニ
1950  W.=0 : WP.=VARPTR(W.) : P.=&HE5F8                  シャライズ
1960  W.=VARPTR(X)    : POKE P.  ,PEEK(WP.) : POKE P.+1,PEEK(WP.+1)
1970  W.=VARPTR(Y)    : POKE P.+2,PEEK(WP.) : POKE P.+3,PEEK(WP.+1)
1980  W.=VARPTR(XW.G) : POKE P.+4,PEEK(WP.) : POKE P.+5,PEEK(WP.+1)
1990  W.=VARPTR(YW.G) : POKE P.+6,PEEK(WP.) : POKE P.+7,PEEK(WP.+1)
2000 'SCREEN 0,0 : DUMMY=USR(CLSM.)
2010 '
2020  BYTE.G=((XW.G+8)¥8)*(YW.G+1)+4 画面が必要とするバイト数の計算
2030  FAC.G=BYTE.G¥2+1                       (但し,白黒モードと考えた時)
2040 'DIM SCRN.G(FAC.G) 画面をget@する時の配列を登録
2050 '
2060  LINE(XW.G+4,0)-(XW.G+4,YW.G),6
2070  LINE(0,YW.G+4)-(XW.G,YW.G+4),6  目盛の表示
2080  FOR I.G=0 TO XW.G STEP 10
2090    LINE(I.G,YW.G+4)-STEP(0,-1),6
2100    IF (I.G MOD 50)=0 THEN LINE(I.G,YW.G+4)-STEP(0,-2),6
2110  NEXT
2120  FOR I.G=0 TO YW.G STEP 5
2130    LINE(XW.G+4,I.G)-STEP(2,0),6
2140    IF (I.G MOD 25)=0 THEN LINE(XW.G+4,I.G)-STEP(4,0),6
2150  NEXT
2160 RETURN
2170 '
2180 *CLS.G                          [e]キーの処理
2190  CLS : INPUT "Right ";A.G$       クロス・カーソルが有効な画面の消去
2200  IF A.G$="y" THEN CLS 2
2210  GOSUB *INFORM.G
```

```
2220 RETURN
2230 '
2240 *PSET.G                       sキーの処理
2250  PSET(X,Y)                    点を打つ
2260  GOSUB *LPSET.G
2270 RETURN
2280 '
2290 *PRESET.G                     rキーの処理
2300  PRESET(X,Y)                  点を消す
2310  GOSUB *LPSET.G
2320 RETURN
2330 '
2340 *LINE.G
2350  LINE(LPX,LPY)-(X,Y) lキーの処理
2360  GOSUB *LPSET.G     線を引く
2370 RETURN
2380 '
2390 *LINEB.G
2400  LINE(LPX,LPY)-(X,Y),,B bキーの処理
2410  GOSUB *LPSET.G         箱の枠を描く
2420 RETURN
2430 '
2440 *LINEBF.G
2450  LINE(LPX,LPY)-(X,Y),,BF fキーの処理
2460  GOSUB *LPSET.G          箱を描く
2470 RETURN
2480 '
2490 *CIRCLE.G
2500  R.G=SQR((X-LPX)^2+4*(Y-LPY)^2)
2510  IF R.G=0 THEN RETURN              cキーの処理
2520  CIRCLE(LPX,LPY),R.G               円を描く
2530 RETURN
2540 '
2550 *LPSET.G
2560  LPX=X : LPY=Y aキーの処理
2570 RETURN         LPを着目座標にする
2580 '
2590 *COLOR.G
2600  C.G=(C.G+1) MOD 8 : COLOR ,,,C.G 〔スペース・バー〕の処理
2610  GOSUB *INFORM.G                   フォア・グラウンド・カラーを変える
2620 RETURN
2630 '
2640 *GET.AT.G
2650  I.G=((ABS(LPX-X)+8)¥8)*(ABS(LPY-Y)+1)*3+4 ⎫ Gキーの処理
2660  J.G=I.G¥2+1                                ⎬ 拾うパターンの大きさをチェック
2670  IF FAC.G<J.G THEN ERROR 7                  ⎭
2680  GET@(X,Y)-(LPX,LPY),SCRN.G ←── パタンのget@
2690  MSG$="Get a picture."
2700  IF BASIC=2 THEN *G.EXIT Rom Basicだとフロッピーディスケットにセーブできないので
2710  CLS : INPUT "Save this picture ";A.G$     *G.EXITへ
2720  IF A.G$="n" THEN *G.EXIT
2730  PRINT : INPUT "Save file name, please. ",FILE.G$
2740  INPUT "Sure ";A.G$
2750  IF A.G$="n" THEN *G.EXIT
2760  BSAVE FILE.G$,VARPTR(SCRN.G(0)),I.G get@したパターンのセーブ
2770  MSG$=MSG$+" and saved."
2780 '
2790 *G.EXIT          ⎫
2800  GOSUB *LPSET.G  ⎬ 冗長? *P.EXITと同じ
2810  GOSUB *INFORM.G ⎭
2820 RETURN
2830 '
2840 *PUT.AT.G ←────── Pキーの処理
2850  IF BASIC=2 THEN *P.SEL ←────── Rom Basicの方は,あしからず…(ロードできません)
2860  CLS : INPUT "Already pattern from file ";A.G$
2870  IF A.G$="n" THEN *P.SEL
2880  PRINT : INPUT "Load file name, please. ",FILE.G$
2890  INPUT "Sure ";A.G$
2900  IF A.G$="n" THEN *P.SEL
2910  BLOAD FILE.G$,VARPTR(SCRN.G(0)) 配列上へパターンをロード
2920 '
2930 *P.SEL
2940  CLS : PRINT "1) PSET     2) PRESET"
```

```
2950  PRINT "3) OR      4) AND"
2960  PRINT "5) XOR" : PRINT
2970  PRINT "Your choice - ";
2980  GOSUB *GET.KEY
2990  ON CH.G-&H30 GOTO *P.PSET,*P.PRESET,*P.OR,*P.AND,*P.XOR
3000 '
3010 *P.PSET    : PUT@(X,Y),SCRN.G,PSET    : GOTO *P.EXIT
3020 *P.PRESET  : PUT@(X,Y),SCRN.G,PRESET  : GOTO *P.EXIT
3030 *P.OR      : PUT@(X,Y),SCRN.G,OR      : GOTO *P.EXIT
3040 *P.AND     : PUT@(X,Y),SCRN.G,AND     : GOTO *P.EXIT
3050 *P.XOR     : PUT@(X,Y),SCRN.G,XOR
3060 '
3070 *P.EXIT
3080  GOSUB *LPSET.G
3090  GOSUB *INFORM.G
3100 RETURN
3110 '
3120 *SAVE.G
3130  CLS : PRINT "Save";
3140  GOSUB *FILE.NAME.G
3150  IF A.G$="n" THEN GOSUB *INFORM.G : RETURN
3160  IF BASIC=2 THEN *CAS.SAVE
3170  FOR I.G=0 TO 2
3180    SCREEN 1,,I.G,2^I.G
3190    GET@(0,0)-(XW.G,YW.G),SCRN.G
3200    SCREEN 0 : LOCATE 4,22 : COLOR 2^I.G
3210    FL.G$=CHR$(34)+FILE.G$+CHR$(I.G+&H30)+CHR$(34)
3220    PRINT USING "Saving from  BANK _##  to  @";I.G+1,FL.G$
3230    BSAVE FILE.G$+CHR$(I.G+&H30),VARPTR(SCRN.G(0)),BYTE.G
3240  NEXT
3250  GOTO *S.EXIT
3260 '
3270 *CAS.SAVE
3280  CONSOLE 0,25 : CLS
3290  FOR I.G=0 TO 2
3300    SCREEN 1,0,I.G,2^I.G
3310    GET@(0,0)-(XW.G,YW.G),SCRN.G
3320    SCREEN ,2 : COLOR 2^I.G : J.G=VARPTR(SCRN.G(0))
3330    FL.G$=CHR$(34)+FILE.G$+CHR$(I.G+&H30)+CHR$(34)
3340    PRINT USING "Saving from  BANK _##  to  @";I.G+1,FL.G$
3350    PRINT USING "address from _&H@ to _&H@ ";HEX$(J.G),HEX$(J.G+BY
TE.G)
3360    MON : PRINT : PRINT
3370  NEXT
3380  SCREEN 0,0 : CLS : CONSOLE 20,5 : CLS
3390 '
3400 *S.EXIT
3410  WINDOW(0,0)-(XW.G,YW.G) : VIEW(0,0)-(XW.G,YW.G)
3420  COLOR 1 : GOSUB *INFORM.G
3430 RETURN
3440 '
3450 *LOAD.G
3460  CLS : PRINT "Load";
3470  GOSUB *FILE.NAME.G
3480  IF A.G$="n" THEN GOSUB *INFORM.G : RETURN
3490  IF BASIC=2 THEN *CAS.LOAD
3500  FOR I.G=0 TO 2
3510    FL.G$=CHR$(34)+FILE.G$+CHR$(I.G+&H30)+CHR$(34)
3520    LOCATE 4,22 : COLOR 2^I.G
3530    PRINT USING "Loading from  @  to  BANK _##";FL.G$,I.G+1
3540    BLOAD FILE.G$+CHR$(I.G+&H30),VARPTR(SCRN.G(0))
3550    SCREEN 1,,I.G,2^I.G
3560    PUT@(0,0),SCRN.G,PSET
3570    SCREEN 0
3580  NEXT
3590  GOTO *L.EXIT
3600 '
3610 *CAS.LOAD
3620  CONSOLE 0,25 : CLS
3630  FOR I.G=0 TO 2
3640    FL.G$=CHR$(34)+FILE.G$+CHR$(I.G+&H30)+CHR$(34)
3650    SCREEN ,2 : COLOR 2^I.G
3660    PRINT USING "Loading from  @  to  BANK _##";FL.G$,I.G+1
```

（注釈）
- 2950～2990: Put@のオプション別の振り分け
- 3120: Ⓢキーの処理
- 3160: Rom Basicの方は，カセット・テープへ…
- 3180: 1つのバンクを選んで…．
- 3190: 白黒モードでget@
- 3230: 画面のある1つのバンクだけセーブ
- 3270: モニタを用いて，カセット・テープへセーブ
- 3300: ある1つのバンクをget@して…
- 3340～3350: 配列の実アドレスを表示
- 3360: モニタに制御を渡す（コントロールⓑで，ここに戻れます）
- 3410～3420: 冗長？＊L.EXITと同じ
- 3450: Ⓛキーの処理
- 3490: Rom Basicの方は，カセット・テープで，どうぞ
- 3540: 画面の1つのバンクをロード
- 3550: 対応するバンクへPut@
- 3610: モニタを用いて，カセット・テープからロード

```
3670     MON : PRINT : PRINT ←―― 制御をモニタへ
3680     SCREEN 1,0,I.G,2^I.G
3690     PUT@(0,0),SCRN.G,PSET ←―― パターンのPut@
3700  NEXT
3710  SCREEN 0,0 : CLS : CONSOLE 20,5 : CLS
3720 '
3730 *L.EXIT
3740  WINDOW(0,0)-(XW.G,YW.G) : VIEW(0,0)-(XW.G,YW.G)
3750  COLOR 1 : GOSUB *INFORM.G
3760 RETURN
3770 '
3780 *FILE.NAME.G
3790  INPUT " file name, please. ",FILE.G$   ファイル・ネーム入力用のセーブ・ロード
3800  INPUT "Sure ";A.G$                      共通のサブルーチン
3810  LOCATE 0,21 : PRINT SPACE$(9)
3820 RETURN
3830 '
3840 *MODE.G                        [m]キーの処理
3850  MODE.G=MODE.G+1
3860  IF MODE.G>3 THEN MODE.G=0     クロス・カーソルのモードを変える
3870  GOSUB *INFORM.G
3880 RETURN
3890 '
3900 *INFORM.G : CLS
3910  PRINT USING "[ x=###  y=### ]";X,Y   グラフィック・エディタの各種情報
3920  PRINT "Fore ground color = ";C.G      を表示する
3930  PRINT "Mode is ";
3940  IF MODE.G=0 THEN PRINT "MOVE ";
3950  IF MODE.G=1 THEN PRINT "PRESET ";
3960  IF MODE.G=2 THEN PRINT "PSET ";
3970  IF MODE.G=3 THEN PRINT "BOLUME LINE ";
3980  PRINT "mode."
3990  PRINT "< Normal <==> Reverse >"
4000  COLOR 2 : PRINT MSG$;
4010  MSG$="" : COLOR 1
4020 RETURN
4030 '
4040 *REVERSE.G                     [>]キーの処理
4050  FOR I.G=0 TO 7                  カラー画面の色を反転!?
4060     COLOR=(I.G,7-I.G)
4070  NEXT
4080 RETURN
4090 '
4100 *RETURN.G                      [<]キーの処理
4110  FOR I.G=0 TO 7                  パレットを通常の状態に…
4120     COLOR=(I.G,I.G)
4130  NEXT
4140 RETURN
4150 '
4160 ' << ﾍﾟｲﾝﾄ ﾒｲﾚｲ ﾉ ｲﾆｼｬﾗｲｽﾞ >>
4170 '
4180 *PAINT.INIT
4190  DIM TILE.G$(4)
4200  GOSUB *MID.INIT  ←―― 中間色ジェネレータのイニシャライズ
4210  GOSUB *MID.DISP  ←―― 中間色の表示
4220  GOSUB *TILE.INIT ←―― タイル・ジェネレータのイニシャライズ
4230 '
4240  RESTORE *T.D
4250  POINT(350,192)
4260  FOR J.G=0 TO 4
4270     FOR I.G=1 TO 12
4280        READ A.G$                                         タイル・パターン
4290        TILE.G$(J.G)=TILE.G$(J.G)+CHR$(VAL("&h"+A.G$))    の読み込み
4300     NEXT
4310     LINE STEP(9,-25)-STEP(20,30),7,B
4320     PAINT(POINT(0)-5,POINT(1)-5),TILE.G$(J.G),7   その表示
4330  NEXT
4340 RETURN
4350 '
4360 *PAINT.G                       [p]キーの処理
4370  CLS : PRINT
4380  PRINT "1) Normal color"
4390  PRINT "2) Middle color"
```

```
4400  PRINT "3) Tiling" : PRINT
4410  PRINT "Your choice - ";
4420  GOSUB *GET.KEY
4430  IF CH.G=13 OR CH.G=&H30 THEN GOSUB *INFORM.G : RETURN  ┐
4440  IF CH.G=&H31 THEN *NORMAL.G                            │選択された
4450  IF CH.G=&H32 THEN *MIDDLE.G                            │番号の解釈
4460  IF CH.G=&H33 THEN *TILING.G                            ┘
4470 GOTO *PAINT.G
4480 '
4490 *NORMAL.G              ←―― 普通のペイント
4500  CLS : PRINT "Normal color-"
4510  INPUT "Paint color, border color(0-7)";P.G,B.G
4520  IF P.G<0 OR P.G>7 THEN GOSUB *INFORM.G : RETURN } 存在しないカラー・コード
4530  IF B.G<0 OR B.G>7 THEN GOSUB *INFORM.G : RETURN } のときは,何もしない
4540  PAINT(X,Y),P.G,B.G
4550  GOSUB *LPSET.G
4560  GOSUB *INFORM.G
4570 RETURN
4580 '
4590 *MIDDLE.G              ←―― 中間色によるペイント
4600  CLS : PRINT "Middle color-"
4610  INPUT "Border color(0-7)";B.G
4620  IF B.G<0 OR B.G>7 THEN GOSUB *INFORM.G : RETURN
4630  PRINT "Select middle color"
4640  CONSOLE 0,25
4650  GOSUB *MID.SEL ←―― 中間色選択用サブルーチン
4660  CONSOLE 20,5 : CLS
4670  PAINT(X,Y),TILE.M$,B.G ←―― ペイントの実行
4680  GOSUB *LPSET.G
4690  GOSUB *INFORM.G
4700 RETURN
4710 '
4720 *TILING.G              ←―― タイリング
4730  CLS : PRINT "Tiling-"
4740  PRINT "Already pattern ?";
4750  GOSUB *GET.KEY : CLS
4760  IF CH.G=&H1B THEN GOSUB *INFORM.G : RETURN
4770  IF CH.G=ASC("y") THEN *TILE.SEL
4780  GOSUB *TILEGEN ←―― タイル・パターンの作成
4790 '
4800 *RETRY.T
4810  COLOR 1 : LOCATE 45,20 : PRINT "0  1  2  3  4" } 作成したタイル・パターンの
4820  LOCATE 0,21 : INPUT "Define number(0-4)";I.G   } 登録
4830  IF I.G<0 OR I.G>4 THEN *EXIT.TS
4840  TILE.G$(I.G)=TILE.T$
4850  SCREEN 0,0 : LINE(360+I.G*24,168)-STEP(18,28),0,BF
4860  PAINT(365+I.G*24,170),TILE.T$,7
4870  WINDOW(0,0)-(XW.G,YW.G) : VIEW(0,0)-(XW.G,YW.G)
4880  LOCATE 0,22 : PRINT "Retry ?";
4890  GOSUB *GET.KEY : PRINT
4900  IF CH.G=ASC("n") THEN CLS : CONSOLE 20,5 : GOTO *TILE.SEL
4910  GOSUB *TILE.PAT : GOTO *RETRY.T
4920 '
4930 *TILE.SEL                  ←―― タイル・パターンを選んでペイント
4940  COLOR 1 : LOCATE 45,20 : PRINT "0  1  2  3  4"
4950  LOCATE 0,20 : INPUT "Border color(0-7)";B.G
4960  IF B.G<0 OR B.G>7 THEN *EXIT.TS   ←―― 存在しない番号なら何もしない
4970  LOCATE 0,21 : INPUT "Pattern number(0-4)";I.G
4980  IF I.G<0 OR I.G>4 THEN *EXIT.TS
4990  PAINT(X,Y),TILE.G$(I.G),B.G
5000  GOSUB *LPSET.G
5010 '
5020 *EXIT.TS
5030  CLS : CONSOLE 20,5 : CLS
5040  GOSUB *INFORM.G
5050 RETURN
5060 '
5070 *T.D
5080  DATA 00,AA,AA,00,AA,AA,00,55,55,00,55,55 ┐
5090  DATA 81,81,81,42,42,42,24,24,24,18,18,18 │
5100  DATA 00,3C,00,00,42,00,00,3C,00,00,00,00 │タイル・パターンの初期データ
5110  DATA 00,00,88,00,00,44,00,00,44,00,00,88 │
5120  DATA 82,82,00,28,28,00,28,28,00,82,82,00 ┘
5130 '
```

```
5140 ' << [HELP] ﾉ ﾄｷﾉ ﾒｯｾｰｼﾞ >>
5150 '
5160 *MSG.BLOCK
5170  HELP OFF : SCREEN ,2 : CONSOLE 0,25 : CLS
5180  PRINT TAB(10);"*** Graphic Editor Command Summary ***" : PRINT
5190  PRINT "[space]  :  ﾌｫｱ･ｸﾞﾗｳﾝﾄﾞ･ｶﾗｰ ﾉ ｺｰﾄﾞﾊﾞﾝｺﾞｳ ｦ ﾋﾄﾂ ﾌﾔｼﾏｽ｡"
5200  PRINT "[  s  ]  :  ﾁｬｸﾓｸｻﾞﾋｮｳ (x,y) ﾆ ﾃﾝ ｦ ｳﾁﾏｽ｡  ( PSET )"
5210  PRINT "              LP ﾊ ｺﾉ ｻﾞﾋｮｳ (x,y) ﾆ ﾅﾘﾏｽ｡"
5220  PRINT "[  r  ]  :  ﾁｬｸﾓｸｻﾞﾋｮｳ (x,y) ﾉ ﾃﾝ ｦ ｹｼﾏｽ｡  ( PRESET )"
5230  PRINT "              LP ﾊ ｺﾉ ｻﾞﾋｮｳ (x,y) ﾆ ﾅﾘﾏｽ｡"
5240  PRINT "[  l  ]  :  ｻﾞﾋｮｳ (lpx,lpy) ｶﾗ ｻﾞﾋｮｳ (x,y) ﾏﾃﾞ ﾁｮｸｾﾝ ｦ ｴｶ
ﾞｷﾏｽ｡  ( LINE )"
5250  PRINT "              LP ﾊ ｻﾞﾋｮｳ (x,y) ﾆ ﾅﾘﾏｽ｡"
5260  PRINT "[  b  ]  :  'b' ｵﾌﾟｼｮﾝ ﾉ LINEﾒｲﾚｲ ﾆ ｿｳﾄｳｼﾏｽ｡"
5270  PRINT "              LP ﾊ ｻﾞﾋｮｳ (x,y) ﾆ ﾅﾘﾏｽ｡"
5280  PRINT "[  f  ]  :  'bf' ｵﾌﾟｼｮﾝ ﾉ LINEﾒｲﾚｲ ﾆ ｿｳﾄｳｼﾏｽ｡"
5290  PRINT "              LP ﾊ ｻﾞﾋｮｳ (x,y) ﾆ ﾅﾘﾏｽ｡"
5300  PRINT "[  c  ]  :  LP ｻﾞﾋｮｳ (lpx,lpy) ｦ ﾁｭｳｼﾝ ﾄ ｽﾙ ｴﾝ ｦ ｴｶﾞｷﾏｽ｡"
5310  PRINT "              ｿｼﾃ ﾊﾝｹｲ ﾊ SQR((x-lpx)^2+4*(y-lpy)^2) ﾃﾞ ｱﾗﾜｻ
ﾚﾏｽ｡"
5320  PRINT "[  p  ]  :  PAINT ﾒｲﾚｲ ﾆ ｿｳﾄｳｼﾏｽ｡ｺﾉ ﾒｲﾚｲ ﾆﾊ 3ｼｭﾙｲ ﾉ ﾓｰﾄﾞ
 ｶﾞ ｱﾘﾏｽ｡"
5330  PRINT "              ﾉｰﾏﾙ､ ﾁｭｳｶﾝｼｮｸ､ ﾀｲﾙ･ﾊﾟﾀｰﾝ ﾉ 3ﾓｰﾄﾞ ﾃﾞｽ｡   ( PA
INT )"
5340  PRINT "              LP ﾊ ｻﾞﾋｮｳ (x,y) ﾆ ﾅﾘﾏｽ｡"
5350  PRINT "[  e  ]  :  ｸﾞﾗﾌｨｯｸ･ｽｸﾘｰﾝ ｦ ｼｮｳｷｮ ｼﾏｽ｡   ( CLS 2 )"
5360  PRINT "[  a  ]  :  LP ｦ ｻﾞﾋｮｳ (lpx,lpy) ｶﾗ ｻﾞﾋｮｳ (x,y) ﾆ ｳﾂｼ､ ｾｯ
ﾄ ｼﾏｽ｡"
5370  PRINT "[  m  ]  :  ｸﾞﾗﾌｨｯｸ･ｴﾃﾞｨﾀｰ ﾉ ﾓｰﾄﾞ ｦ ｶｴﾏｽ｡"
5380  PRINT "[ S/L ]  :  ｸﾞﾗﾌｨｯｸ･ﾃﾞｰﾀ ｦ ｾｰﾌﾞ / ﾛｰﾄﾞ ｼﾏｽ｡"
5390  PRINT "[ G/P ]  :  ｸﾞﾗﾌｨｯｸ･ﾃﾞｰﾀ ｦ GET@ / PUT@ ｼﾏｽ｡"
5400  PRINT "[  >  ]  :  ｸﾞﾗﾌｨｯｸ･ｽｸﾘｰﾝ ﾉ ｲﾛ ｦ ﾊﾝﾃﾝ ｼﾏｽ｡"
5410  PRINT "[  <  ]  :  ｸﾞﾗﾌｨｯｸ･ｽｸﾘｰﾝ ﾉ ｲﾛ ｦ ﾉｰﾏﾙ ﾆ ﾓﾄﾞｼﾏｽ｡";
5420 '
5430 *WAIT.LOOP.MB : IF INKEY$="" THEN *WAIT.LOOP.MB
5440  CLS : CONSOLE 20,5 : CLS
5450  SCREEN ,0
5460  GOSUB *INFORM.G : HELP ON
5470 RETURN
5480 '
5490 ' << ｴﾗｰ ﾄﾗｯﾌﾟ >>
5500 '
5510 *TRAP
5520  LOCATE 0,24 : PRINT SPACE$(20);
5530 '
5540  IF ERR= 5 THEN MSG$="Illegal function call"
5550  IF ERR= 7 THEN MSG$="Out of memory"
5560  IF ERR=52 THEN MSG$="Bad file number"
5570  IF ERR=53 THEN MSG$="File not found"
5580  IF ERR=56 THEN MSG$="Bad file name"
5590  IF ERR=61 THEN MSG$="File write protected"
5600  IF ERR=62 THEN MSG$="Disk offline"
5610  IF ERR=64 THEN MSG$="Disk I/O error"
5620  IF ERR=68 THEN MSG$="Disk full"
5630  IF ERR=69 THEN MSG$="Bad allocaton table"
5640  IF ERR=70 THEN MSG$="Bad drive number"
5650  IF ERR=72 THEN MSG$="Deleted record"
5660 '
5670  BEEP : RESUME *TRAP.1
5680 '
5690 *TRAP.1
5700  GOSUB *INFORM.G
5710 RETURN
```

ここでMSG$に代入された文字列は＊INFORM.Gで表示される

〈中間色ジェネレータ入る〉
〈タイル・パターン・ジェネレータ入る〉

☆プログラムの説明と使い方

メイン・ルーチンは，1030行から1060行までの4行だけです．

このプログラムは，機械語のパッケージがメモリ上に常駐していることを前提としていますの

で，そのプログラムを破壊しないように，CLEAR 文でメモリの使用領域の上限を設定します．次の行に示されている＊INITIALIZE というサブルーチンは，当然のことながらGエディタの初期化を行うものです．この初期化はツールとの兼ね合いもあり，重要な点をいくつか含んでいます．その最もポイントとなるものは，機械語によるツールとリンクです．

機械語のパッケージとのパラメータのやり取りは，整数型を使う必要があるため，DEFINT A－Zを実行してあります．後に，実数型の変数が必要となった場合は，〝！〞をその変数名の末尾に付けるなどして，解決してください．ちなみに，Gエディタでは実数型の変数は1つしか使用していません（＊MAIN．LOOP の中の R！）．

もう1つは，機械語のパッケージをユーザー関数として呼び出しているので，DEF USR＝&HE400を実行します．さらに，Gエディタと機械語のパッケージが，共通に用いる変数領域があるので，それを明確にし，機械語のパッケージに知らせる必要があります．実行しているのはプログラムの＊GR．INIT です．

機械語のパッケージが知る必要のあるものは，着目している点の座標，クロス・カーソルが移動することのできる範囲の最大値を保持している変数が登録されているメモリ上の実アドレスです．これらは，最初に変数領域に登録され，後にその位置が変わらないように，DEFINT A-Z のすぐ後で，代入文を実行しています．

Gエディタでは，クロス・カーソルの移動するｘ，ｙ軸各方向の最大値を保持する変数をXW．G，YW．G，クロス・カーソルの着目している点の座標を保持する変数をX，Yとして，＊GR．INIT 内で変数領域に登録しています（と言っても，単なる代入文ですが……）．この後に続く POKE 文の数行が，X，Y，XW．G，YW．Gの登録されたアドレスを，機械語のパッケージに知らせています．また，POKE 文の直前の代入文は，単語に意味を持たせているにすぎません．プログラムの中で，ユーザー関数を用いた時，

```
DUMMY＝USR（パラメータ）
```

という形をとりますが，パラメータが単なる数値では，何の機能が実行されるのか，プログラムを見ただけでは分かりにくくなってしまいます．やはり；プログラムの分かり易さを考えると，このような形にしておいた方が無難だと思います．詳細については，4．3の〝機械語によるツール〞に記してありますので，そちらを参照してください．

次に，ツールとの兼ね合いで重要な初期化は，＊PAINT．INIT で行っています．機械語以外のツールで用いているのは，＊MID．COLOR と ＊TILEGEN ですが，この2つはどちらも，PAINT 文に関連しているので，イニシャライズをここで行います．

最初にストリング変数の配列，TILE．G$（ ）を定義していますが，これはGエディタの保持するタイル・パターンで，これらには，＊T．D 以降に示されているデータが，ストリングの形で代入されます．また，後に定義し直すことも可能ですが，そのことについては，使い方の項で記すことにします．

次に続く3行が，ツールのイニシャライズ・ルーチンを呼び出している所で，中間色の方は，ここで表示をするルーチンも呼び出しています．それぞれのツールの初期化についての詳細は，4．2．1や4．2．2に示した通りです．タイル・パターン（＊T．D 以降のデータ）は，すべてのツールのイニシャライズが終わった後に表示しています．

以上のように，無事にイニシャライズを終えますと，今度は，キーボードからのコマンド入力と，その解析，コマンドによる画面上に影響する各サブルーチンの制御をするルーチン——＊MAIN．LOOPが実行される番です．

キーボードからのコマンド入力は，＊IN．KEY というルーチンで受け付けており，その後，INSTR 関数を用いることによってその解析を行っています．＊MAIN．LOOP 内において，キーボードからの入力をINKEY$ 関数で統一しているのは，機械語パッケージが，INKEY$ 関数で受けた命令を実行するからです．機械語のパッケージは，キーボードの右側に配置されているテン・キーによって，クロス・カーソルを移動する処理を受け持っています．

もう1つ，キーボードからの入力を司るサブルーチンが，＊GET．KEY です．このサブルーチンは，Gエディタの他の部分でかなり用いており，特にGエディタとの会話部分において，返答に間違いを起こしてもあまり重要でない部分に多く応用しています．

＊MAIN．LOOP では，大きく2つの流れ——つまり，クロス・カーソルの移動に関するものと，画面上の制御（点を打つ，線を引く，塗る etc.）に関するものに分かれています．

クロス・カーソルの移動に関する処理は，＊CURSOR．G 以降で行い，Gエディタのモード（[M モ]のキーで変化し，MODE．G がモードに対する値を保持している）によって，次の4つの動作をします．つまり，MODE．G の値が，

0の場合：単純に，テン・キーの指示に従って移動する．（図6）この時シフト・キーを押しているとカーソルが速く動く．

1の場合：着目している所に，バック・グラウンド・カラーで示されている色（黒）の点を打つ（言い換えれば，PRESET 命令を行う）．その後，0の場合と同様に移動する．

2の場合：着目している所に，フォア・グラウンド・カラー（スペース・バーを打つことによって変化する）で示されている色の点を打つ（言い換えれば，PSET 命令を行う）．その後，0と同様に移動する．

3の場合：名付けて，VOLUME LINE．クロス・カーソルを移動しないと，インクがしみ出すような効果を得ることができ，一風変わったタッチのパターンを描くことができる．

MODE．G が3の時 VOLUME LINE は，単に，クロス・カーソルが移動しない時に，半径を増しながら円を描いているだけです．クロス・カーソルを表示したまま円を描くと，その円とクロス・カーソルの交わった部分は，フォア・グラウンド・カラーとは異なった色になってしまいます．この理由で，このモードの時のみクロス・カーソルがちらつきます．また，移動したか，しないかは，BASIC でチェックしても良いのですが，機械語のパッケージでクロス・カーソルの

移動に関するルーチンを呼び出すユーザー関数は，移動した時のみ255という値を返してくれるので，これを利用して，チェックしています（移動していなければ，0を返します）．

図6　テンキーとクロス・カーソルの動き

画面上の制御に関する処理は，そのコマンド別に，ON VEC．GOSUB～で，振り分けています．振り分けられてからの各サブルーチンは，単純なものばかりですから，ちょっと見慣れない操作のみ詳しく触れ，その他は，キーと機能だけの説明にとどめておきます．また，以下に述べるコマンドの説明の要約は，Gエディタを実行中に，HELP キーを押せば，画面上に表示されます．その処理をしているのが，リスト8の＊MSG．BLOCKです．この表示されている状態から，もとの状態に戻るのに，何か適当なキー（大きくて，たたきやすいスペース・バーや，リターン・キーなど）を押してください．では，キーの説明に入りたいと思います．

スペース・バーを押すと，フォア・グラウンド・カラーを1つインクリメント（加算）します．Gエディタを動かしたばかりの状態では，フォア・グラウンド・カラーは，白（カラー・コード7）に初期化されています．また，Gエディタを実行すると分かりますが，この情報は，常に画面の下に表示されています．A チ を押すと，LPを現在着目している座標に変更します．リスト8の中では，着目している座標を保持している変数がX，Y，LPの座標を保持している変数が，LPX，LPYです．従って，＊LPSET．Gで行っているように，単に着目している座標をLPの座標に代入するだけの命令です．初期状態では，XとLPX，YとLPYは，全く同じ値を持っています．これから紹介する他のコマンドは，ほとんどこれら2つの点に関係して作用し，実行後，LPの座標値は着目座標の値になります．

S ト キーは，着目している座標に，フォア・グラウンド・カラーの示す色の点を打ちます．要するに，

```
PSET (X, Y)
```

と同じ作用をします．

R ス キーは S ト キーとは逆に，着目している座標にバック・グラウンド・カラーの示す色（黒）の点を打ちます．これも，

```
PRESET (X, Y)
```

と同じ動きをするのは，言うまでもありません．

L リ キーは，LPと着目座標との間に，フォア・グラウンド・カラーで示された色の線を引きま

す．この命令もLPが，新しい着目座標に移っていきますので，短い線を少しずつ描けば——つまり，クロス・カーソルを少しずらして L リ キーを押す，といったことを繰り返せば，ある程度の曲線も表現することができます．

B コ キーを押した時は，L リ キーを押した時，描かれる線を対角線とする四角形（長方形）を描きます．F ハ キーを押した時も同様に四角形を描きますが，こちらの方は四角形の中が塗りつぶされます．これらの実行は，LINE 文において，BやBF のオプションを付けたものと同様な結果を得られます．プログラムでは，*LINEB．G，*LINEBF．Gのサブルーチンで処理しています．

C ソ キーは，Gエディタに円を描かせるコマンドです．中心の座標は，LPの示す点で，半径は，着目している座標で決定されます（着目している点を通るような円を描く）．この命令では，実行してもLPが保持されていて変化しないので，同心円が容易に描けます（図7）．

P セ キーは，各種のPAINT命令（①単純に色を塗る，②パッケージのサブルーチンを用いて中間色やタイリングによって色を塗る）を実行するモードに入ります．その後は，表示される文に従って，会話形式に答えていってください．また，それぞれのサブルーチンから，色を塗る前に脱け出したいときは，ボーダー・カラーに対して，0より小さい値や，7より大きい値を指定してください．各サブルーチンに入る前に，PAINTのモードから脱け出す時は，リターン・キーもしくは0を入力してください．PAINTは，言うまでもなく着目している座標から行います．

E イ キーを押すと，画面の消去（VIEWで指定した内側のみ）をします．但し，キーを押したとたん消し始める訳ではありませんから，間違って押してしまっても慌てなくても大丈夫です．もう一度だけ，Gエディタが，本当に消していいのか念を押してきますので，これに対して〝y〟（Yes）または，〝n〟（No）で答えてください．

M モ キーは，既に説明した様に，エディタのモードを変化させるコマンドです．1回押す度に，モードはインクリメントされていき，0→1→2→3→0→1……の様に変化します．これは，フォア・グラウンド・カラーと同様に，今どのモード（MOVE，PRESET，PSET，VOLUME LINEのいずれか）にいるのかという情報は，画面の下に表示されています．

SHIFT ＋ G キ キーは，N88-BASICのGET＠文と同じ働きで，配列に画面上のパターンを拾います．拾ってくるパターンの大きさは，カーソルで指定しますが，B コ キーを押した時に描かれる枠の内側（枠を含む）と同じ大きさです（図8(a)）．つまり，LPの示す点と着目座標の示す点を結んだ線を対角線とする四角形を拾う訳です．さらに，拾ったパターンの収まっている配列の中身のデータを外部の記憶装置に出力して，保存（セーブ）することもできます．しかし，残念ながらROM BASICにおいては，画面全体をテープにセーブすることはできるのですが，SHIFT ＋ G キ で拾ったパターンは，セーブすることができません．GETした後は，そのことが実行されていれば，画面の下に赤い文字で，

Get a picture.

と表示されます．もし，配列に収まらない大きさのパターンを指定した時は，やはり画面の下に，

Out of memory.

の表示が現われますから，もう少し小さい領域にしてください．

SHIFT ＋ Pセ キーは，PUT@文と同じ機能です．但し，配列上にあるパターンのみならず，先の SHIFT ＋ Gキ キーの際に外部の記憶装置に保存したパターンも，配列にロードしてきて，それを画面に置くこともできます．さらにPUTする時のモード（PSET，AND，ORなど）も選ぶことができます．通常は，PSETで十分ですが，すでに画面にあるパターンと重ねる時は，ORやXORを使います（図8(b)）．この時，当然ロードする前に配列にあったパターンは，失なわれることを忘れないでください．また，ROM BASICの場合は，テープに対して画面の一部をセーブできませんので，ロードもできません．パターンをPUTし始めるのは，着目座標からで画面からはみ出してしまう位置でこの命令を入力すると，何もせずにコマンドの入力持ち状態になり，画面の下に，

Illegal function call.

と表示されます．

上記の SHIFT ＋ Gキ，SHIFT ＋ Pセ キーによるコマンドを処理するサブルーチン（*GET．AT．G，*PUT．AT．G）の中で，初めて画面のセーブ，ロードの考え方を取り入れているの

図7　円の描き方

(a)　GETの仕方　　(b)　PUTの仕方

図8　GET／PUTの仕方

で，そのことについて少々触れておきましょう．

まず，セーブの方法を考えたとき，Disk-BASIC の場合ですと，スピードの点で GET@文によって配列に得たパターンを BSAVE 文を用いて，フロッピー・ディスク（ディスケット）上にセーブするのが一番速いようです．カセット・テープにセーブするなら，同様な理由でモニタの持っている機能でメモリの内容をカセット・テープに対して書き出すコマンドのWを用いるのが，良い方法と考えられます．

BSAVE を用いる時，注意しなければならないことは，セーブを開始するメモリのアドレスをパラメータとして与える際，その直前で今まで用いなかったような他の変数に代入して，それをパラメータにすると，その実際のアドレスが，代入された変数の登録によってずれてしまい，プログラムとしてうまく動作しなくなるということです．従って，これを解決するには，配列のアドレスを他の変数に代入しなければいい訳ですから，リストに示してある様に，BSAVE 文のパラメータの部分に，直接 VARPTR 関数を用いると良いでしょう．また，これ以外のパラメータに用いる変数なども，登録済みになるように，工夫しておいた方が，無難だと思います．例えば，イニシャライズの時に，プログラム中で用いる変数をすべて登録（代入文を行う）しておけば，安心でしょう．

セーブする時のバイト数の計算は，もう御馴染みのGET＠文の時に使った計算式

((横のドット数＋７)￥８) ＊Ｍ＊縦のドット数＋４

但し，Mは，カラー・モードの時に3，白黒モードの時に1

を用いてください．プログラムでは，＊GET．AT．G の一番最初で I．G に代入しています．また，上に示した式とは，少し形が違っていると感じた方がいるかもしれませんが，ドット数を求めるには，差の絶対値に1を加えなければいけないのです．例えば，x 軸方向の座標で，3と7という値を与えられたとしてみましょう．そこで，この2つの値の差の絶対値を求めてみますと4になりますが，実際に図で示してみると5ドットあることが分かります(図9)．これは，単に差の絶対値をとっただけでは，与えられた2つの値の点を片方しか数えていないことになるので，1を加えることによって残りの点も数えたことにしている訳です．

……1　2　3　4　5　6　7　8……

図9

次に，ロードの方法もセーブと同様に考えることができるので，Disk-BASIC の場合ですと BLOAD，ROM BASIC の場合ですと，モニタのＲコマンドを用いてロードするのが良いようです．BLOAD 文を用いる時も，考え方や扱う上で注意などは，BSAVE 文と全く同じですから，説明を繰り返すことはやめておきます．リスト8では，＊PUT．AT．G の所で処理していますから，そこを参考にしてください．

SHIFT ＋ G キーでは，指定した範囲のパターンに対してしか，セーブ／ロードを行うこと

ができませんでしたが，画面全体（クロス・カーソルが，移動する範囲）をセーブ／ロードすることができるコマンドも用意しました．

SHIFT ＋ S キーは，セーブをGエディタに指示します．*SAVE．G では，SHIFT ＋ G キーで行えるセーブと同様にDisk-BASIC の場合，BSAVE 文を用いていますが，さらに ROM BASIC のために，モニタのWコマンドを用いてテープにセーブできるようになっており，PC-8801の3つの各プレーンごとにファイルを作成していきます．従って，出来上がるファイルは3つになり，ファイル名の末尾に〝0〟～〝2〟の数字を付けることによって区別されます．例えば，ファイル名をGエディタに聞かれた時に〝Gfile〟と入力したとすれば，ディスケットには，〝Gfile0〟，〝Gfile 1〟，〝Gfile 2〟の3つのファイルができることになります．これらは，BSAVE 文によってセーブされたものなので，機械語のプログラムと見なされ，ファイル名と拡張ファイル名が*（アスタリスク）で区切られています．但し，テープにセーブする時には，異なったファイル名（一応，ファイル名を画面に表示する）をモニタのWコマンドに与えてください．この時に，開始アドレス，終了アドレスは，画面に表示された値を用いてください．

SHIFT ＋ L キーは，ロードです．このときGエディタに知らせるファイル名は，末尾に数字を付けない形で入力します(但し，ここでいう数字とは，プレーンの番号のことです)．先程の例で示したファイルをロードする時も，ファイル名を聞かれたら，〝Gfile〟と入力するだけで結構です．もちろん，ロードも各プレーンごとに，メモリ上（配列の内部）にパターンをロードし，それを PUT@によって画面上に置いていきます．ROM BASIC を使っている人は，テープからモニタのRコマンドによってロードをしてください．画面全体のセーブとロードを，モニタで行った人(ROM BASIC を使用している人は)は，CTRL ＋ B によってモニタから脱け出してください．

ROM BASIC を使っている方も，セーブする時とロードする時の配列の位置がずれないように，イニシャライズで，プログラム中で用いる変数をすべて登録されるように，代入文を実行しておいてください．

さらに，付加的な機能として，SHIFT ＋ < ，SHIFT ＋ > キーを用意しました．この2つのコマンドは，画面の下のコメントに出ている様に，

<Normal <==> Reverse>

の矢印（〝<〟と〝>〟）の方向の機能と一致しています．

SHIFT ＋ > (Reverse)キーは，画面の色をパレットを用いて反転させます．反転の方法は，*REVERSE．G に示されているように，7からカラー・コードを引いた値をパレットに与える様にしています．白地に黒い線で描いたパターンを得たい時などに，白地に黒い線を描くより，普通の状態で，白い線を用いてパターンを描いた方が見易いので，この様なときに，Reverse の機能を用いることができます．

逆に，Reverse で反転させたパターンのパレットを通常の状態に戻す機能が，SHIFT ＋ <

(Normal)キーで，リスト8では，＊RETURN．Gで処理しています．もうお分かりの様に，この2つのコマンド（SHIFT ＋ ＞ ， ＜）は，写真のネガとポジのように対になっている訳です．

尚，以上に示したキーボードからの入力操作は，CAPSがoffの状態，つまり，通常の入力で小文字が打てる状態で行ってください．

4.3 機械語によるツール(Glib)

N_{88}-BASICにはグラフィックスのための強力なステートメントや関数が多くあり，これを使いこなす事で実に多彩な事が実現できます．ところで，その際不満を感じる事はありませんか？例えば，グラフィックの画面の消去（CLS 2）の実行が遅いとか，グラフィック画面のスクロール（ROLL）はDisk-BASICでしか使えないなどです．CLS 2もSCREEN，3を実行してから行えばいくぶん速くなりますし，ROLLもGET＠を使って実現できない事もありませんが，BASICで記述する限り速度の向上は望めません．そこで，グラフィックのいくつかは機能を選び，それぞれ機械語で記述したサブルーチン・ライブラリ(リスト15)，名付けて〝Glib〟を作成しました．

このライブラリはBASICで簡単に使える様，USR関数で呼び出します．このライブラリのソース・リストについてもここで説明しますので，PC-8801のグラフィック関係のI／Oポートを機械語で使う場合などの参考になると思います(但し，ソース・リストは長大なため巻末のAPPENDIXに掲載します)．但し，以下に紹介する各チェック・プログラムを実行する前に，必ずリスト15の機械語プログラム（Glib）を入力しておいてください．

4.3.1 Glib

このライブラリには18の機能があり，これらはUSR関数で呼び出す様になっています．USR関数ではパラメータを1つ与える事ができますから，このパラメータで機能の選択をします．例えば，グラフィックの全画面クリアはパラメータ0で，

```
DUMY=USR(0)
```

とする事で良いのです．まずは，このパラメータについて，どの様な機能があるかを説明しましょう．

a) CLS

画面のクリアには次の3種類があります．

```
USR(0)……SCREEN 0の CLS 2
USR(1)……SCREEN 1の CLS 2
USR(2)……SCREEN 2の CLS 2
```

このうちUSR(1)は，白黒640×200ドット3画面モードですから，どの画面を消去するかを指

定しなければなりません．

b) ROLL

ROLLは，グラフィック画面のスクロールで，上下左右4方向，さらに，それぞれについてSCREEN 0と2の場合の2つがあり，合計で8つの機能があります．

USR(3)……上スクロール（SCREEN 0）
USR(4)……上スクロール（SCREEN 2）
USR(5)……下スクロール（SCREEN 0）
URS(6)……下スクロール（SCREEN 2）
USR(7)……左スクロール（SCREEN 0）
USR(8)……左スクロール（SCREEN 2）
USR(9)……右スクロール（SCREEN 0）
USR(10)……右スクロール（SCREEN 2）

BASICのROLL文は，この内USR(3)とUSR(4)に相当します．

c) パターン反転

これは配列内にGETしたグラフィック・パターンを回転させたり，反転させるためのもので，次の3種類があります．

USR(11)……対称軸反転
USR(12)……X方向反転
USR(13)……Y方向反転

d) クロス・カーソル

これは，画面上をドット単位で指定するためのグラフィック・カーソルを操作するものです．USR(14)でカーソルを表示させ，USR(15)でカーソルの移動と位置の読み取り，そして再びUSR(14)でカーソルを消します．

e) 漢字1行PUT

5章では漢字ワード・プロセッサを紹介しますが，少しでも実行速度を向上させるために，漢字表示の一部分を機械語でサポートしました．USR(16)とUSR(17)がこのためのもので，それぞれSCREENが1と2の時の場合になっています．この機能は指定した行に1行分漢字を表示させますが，漢字ワード・プロセッサの内部データを受け取り表示する様になっているので，残念

ながら汎用的な機能ではありません.

以上, USR関数のパラメータと機能について簡単に説明しましたが, このパラメータは必ず整数形にして用います. このライブラリのエントリ(入口)では, まず最初にUSR関数の値を受け取り, それに応じたルーチンへ飛んでいますが, この時にパラメータが整数型でない時は何もせず戻る様になっています.

また呼び出す機能を間違えた時, 機能によってはシステムを破壊してしまう事もあります. そこであらかじめ変数に値を代入しておくと良いでしょう.

さて, USR関数の引数で機能を選択しますが, この他にもパラメータが必要な場合があります. 例えば上スクロールの場合には何ライン分スクロールするかを指定します. このためのパラメータ領域はE5F8H(末尾にHがあるものは16進数を示す)からE5FFHになっていますが, 直接ここに値を書き込んではいけません. ライブラリではこの領域を2byteずつ4つのパラメータとし, その値をアドレスとして間接参照しているのです. なぜこの様な事をするかと言うと, POKE文やPEEK関数を使ってパラメータのやり取りをするのは意外と面倒で, 実行速度の面からも無視できないからです.

次のプログラムでは, 20行でARG 0. という変数のアドレスを求め, 30行, 40行でパラメータ領域にその値を書いていますが, こうする事で, ライブラリは間接参照でARG 0. の値をパラメータとするのです. ですから, 例えば2ライン分スクロールさせようという場合には,

```
ARG 0. = 2 : DUMY=USR( 3 )
```

とするだけで済みます.

但し, このARG 0. という変数の型に注意してください. 機能を選択するUSR関数の引数に限らず, この機械語ライブラリではパラメータの型はすべて決まっていて, 整数型でなければなりません. このプログラムの場合には, DEFINT文を実行しているから良いのですが, DEF INTを使えない場合は, 必ず%を使って整数型にしてください.

リスト8

```
10 CLEAR ,&HE3FF : DEFINT A-Z : DEF USR=&HE400
20 ARG0.=0 : W=VARPTR(ARG0.)
30 POKE &HE5F8 , PEEK (VARPTR(W))
40 POKE &HE5F9 , PEEK (VARPTR(W)+1)
```

ところで, このライブラリはUSR関数を使っていますが, USR関数で機械語プログラムを使う時には, まずCLEAR文で機械語領域を確保し, その上でDEF USRにより機械語ルーチンのエントリを宣言しなければなりません.

このライブラリはE400HからE5FFHまで512byteをしめていて, エントリはE400Hとなっています. そこでライブラリを使っているBASICプログラムでは, 必ず, 次の2つの文を最初に実行しなければなりません.

```
CLEAR , &HE3FF
DEF USR=&HE400
```

さて今まで説明した事をまとめて，一般的なイニシャライズ・プログラムを次に挙げます．なお，以後の各機能の説明では，このプログラムによってイニシャライズしたものと仮定します．

リスト9

```
100 '
110 ' G-lib ｲﾆｼｬﾗｲｽﾞ ﾙｰﾁﾝ
120 '
130 CLEAR ,&HE3FF
140 DEF USR=&HE400 : PARAM.%=&HE5F8
150 ARG0.%=0 : ARG1.%=0 : ARG2.%=0 : ARG3.%=0
160 W.%=0 : WP.%=VARPTR(W.%)
170 W.%=VARPTR(ARG0.%):POKE PARAM.%  ,PEEK(WP.%):POKE PARAM.%+1,PEEK(W
P.%+1)
180 W.%=VARPTR(ARG1.%):POKE PARAM.%+2,PEEK(WP.%):POKE PARAM.%+3,PEEK(W
P.%+1)
190 W.%=VARPTR(ARG2.%):POKE PARAM.%+4,PEEK(WP.%):POKE PARAM.%+5,PEEK(W
P.%+1)
200 W.%=VARPTR(ARG3.%):POKE PARAM.%+6,PEEK(WP.%):POKE PARAM.%+7,PEEK(W
P.%+1)
210 '
220 CLS.%=0:CLSS.%=1:CLSH.%=2 : ' cls
230 ROLUP.%=3:ROLUPH.%=4 : ' roll up
240 ROLDW.%=5:ROLDWH.%=6 : ' roll down
250 ROLLE.%=7:ROLLEH.%=8 : ' roll left
260 ROLRI.%=9:ROLRIH.%=10: ' roll right
270 PROT.%=11:PROTX.%=12:PROTY.%=13 : ' pattern rotate
280 GCREV.%=14:GCMOVE.%=15 : ' G-cursor
290 KPUT.%=16:KPUTH.%=17 : ' kanji put
```

このプログラムでは，パラメータ受け渡しのための変数として ARG 0.，ARG 1.，ARG 2.，ARG3. の 4 つを定めています．また，機能を選択する USR 関数の引数も，定数宣言の様に定めていますので，例えば，グラフィック画面の下への10ライン分のスクロールは，

```
ARG0.%=10
DUMY=USR(ROLDW.%)
```

となり，かなり分かり易くなると思います．このプログラムで変数の最後に「.」を付けていますが，これはライブラリを使うための変数の意味を強調するためのものです．

以上で，ライブラリを呼ぶ上でのイニシャライズの仕方が理解できたと思いますので，各機能について，グラフィックスに関係するＩ／Ｏポートの説明を交えながら，さらに詳しく説明しましょう．但し，以降の解説は本書中のプログラムの実行に関し，必ずしも必要な部分でありませんから，機械語の知識のない方は読み飛ばしても結構です．また，実際のソース・リストは巻末の APPENDIX に掲載されています．

4.3.2 CLS

まず，次の図を見てください．これは PC-8801のメモリのバンク構造です．PC-8801で使っている CPU の μPD780C-1（Z-80A相当）はアドレス空間が64K byte ですので，グラフィック VRAM, BASIC ROM等を含めて184 Kbyte のメモリを扱うために，この様な複雑な構造になっ

メイン・バンク
0000
BASIC
プログラム
8000
テキスト・ウインドウ
8400
C000
システム作業領域等
E600
テキスト・VRAM
ジャンプ・テーブルなど
FFFF

BASIC
ROM

N-BASIC
ROM
6000
N88 モニタROM
7FFF

N88-BASIC ROM
(グラフィック関係)

グラフィック・バンク
G-VRAM#1
(青)
G-VRAM#2
(赤)
G-VRAM#3
(緑)

図10 PC-8801のバンク構造

ています(第1章参照).

さて,ここでは画面クリアのため,まずグラフィック・バンクをセレクトしなければなりません.そのためのポートはI/Oの5CHから5FHまでにあり,次の表の様になっています.

5CHから5EHにデータを書き込む事で,それぞれグラフィック・バンクがセレクトされ,C000HからFFFFHに現れます.そして5FHに書き込む事で再びメイン・バンクへと切り替わりますが,この時切り替わるのは,C000HからFFFFHの間だけです.

また,現在どのバンクになっているかは,5CHから5FHの内容を読み出す事で知る事ができます.下位3bitが最下位bitからバンク1,2,3のステータス・ビットで,1の時に,そのバンクはセレクトされています.また2つ以上のバンクが同時にセレクトされる事はなく,そして3bit共0の時には,メイン・バンクがセレクトされていることを示します.

これで,一応バンクの切り替えは出来るのですが,その際には充分注意しなければなりません.というのはメイン・バンクのこの領域には,BASICのジャンプ・テーブルや割り込みベクタ,及び処理ルーチン,さらにワーク・エリアなど重要なデータやプログラムがあるからです.しかも,N_{88}-BASICでは割り込みを多用していますので,バンクを切り替える時には,DI(割り込み禁止命令)を行って割り込みを禁止し,グラフィック操作を終えた後,EI(割り込み許可命令)を行う前に必ずメイン・バンクに再び切り替えておかなければなりません.さもないとグラフィック・データをプログラムとして実行してしまいます.

I/Oアドレス	Write	Read
5CH	バンク1をセレクト	現在セレクトされているバンクを下位3bitで表す.
5DH	バンク2をセレクト	
5EH	バンク3をセレクト	
5FH	メイン・バンクをセレクト	

表3 グラフィック・バンク・セレクトI/O

b_7 … b_3	b_2	b_1	b_0
(未使用)	G3	G2	G1

表4 バンク セレクト・ポート・Stautus

I/Oアドレス	Read	Write
70H	オフセット・レジスタ Read	オフセット・レジスタ Write
78H		increment オフセット・レジスタ

表5 テキスト・ウィンドウ オフセット・レジスタ

ところで, このライブラリはE400HからE5FFHに置かれていると前に述べましたが, このままでは, グラフィック・バンクに切り替えたとたんに暴走してしまいます. 実は, このライブラリは2つに分かれていて, もう1つは, 8000Hから83FFHにあるのです.

N_{88}-BASICモードにおいて, この領域は, テキスト・ウィンドウと呼ばれる0000Hから7FFFHまでのBASIC ROMの裏にあたるテキスト領域をアクセスするためのウィンドウとなっています.

N_{88}-BASICモードでは, 通常0000Hから7FFFHは, ROMがセレクトされており, この裏バンクにあるBASICテキスト領域をアクセスするために, 8000Hから83FFHまでへのアクセスは, ハードウェアのアドレス変換ロジックによりオフセットを加え, メイン・バンクの0000Hから7FFFHをアクセスするように変えているのです.

さて, この時メイン・バンクの8000Hから83FFHは, どうなっているでしょう. N_{88}-BASICでは, このテキスト・ウィンドウの機能をBASICテキストのためだけに使っていて, 実はメイン・バンクのこの領域は, 全く使っていないのです. そこで本ライブラリではグラフィック・バンクを操作するルーチンをここに置くことにしました.

CLSのプログラムもこの領域にありますが, このCLSはBASICのそれとは全く別のものです. BASICのCLSでは, ビューポート内だけを消すために, 実際に行っている事はLINE文のBFオプションの時と, ほとんど同じなのです. そこで実行速度を向上させるために, このライブラリでは, ビューポートなどを考慮せず画面すべてを消していますが, これにより実行速度は, USR(0)ではSCREEN, 3の時で約0.4秒と格段に早くなりました.

0000
N88-BASIC
ROM
8000
テキスト・ウィンドウ
8400
メインバンク
0000
XXX0
XXXF
7FFF
BASIC テキスト 領域

図11 テキスト・ウィンドウ

15 8 7 0
1 0 0 0 0 0 A9
+
15 8 7 0
O7 O0 0 0 0 0 0 0 0 0 ……… オフセット・レジスタの値
⇩
アクセス・アドレス

図12 テキスト・ウィンドウ・アドレス変換

また，前にも述べましたが，3種類の画面クリア，USR(0)～USR(2)があり，それぞれのUSR関数のパラメータは，SCREEN 文の第1パラメータに対応しています．そして，この内USR(1)にはどのバンクを消去するのかを指定するパラメータが必要で，それぞれバンク1から3に対応して0から2を与えます．間違った値（例えば4など）を与えてもチェックしませんから，画面

1 byteずらしてブロック転送
転送前 0
⇩
1byte転送後 0 0
⇩
2byte転送後 0 0 0
⇩
転送終了 0 0 0 0 …………… ……… 0 0
⇩

図13 ブロック転送によるメモリ

USRの引数	パラメータ	機　　能
0	な　し	SCREEN 0 の時の CLS 2
1	バンクを指定	SCREEN 1 の時の CLS 2
2	な　し	SCREEN 2 の時の CLS 2

表6　画面消去のパラメータと機能

クリアをしようとしますが，その結果は確実に暴走しますので注意してください．

このCLSのプログラムでは，Z-80のブロック転送命令(LDIR)を使って，グラフィックVRAMに0を書き込んでいますので，実行後は白黒モード(SCREEN 0, SCREEN 2)の時にはCOLOR文で指定したバック・グラウンド・カラーになります．

リスト10

```
100 '
110 '  graphic library CLS test program
120 '
130 CLEAR ,&HE3FF : DEFINT A-Z
140 DEF USR=&HE400 : PARAM.=&HE5F8
150 ARG0.=0
160 W.=0 : WP.=VARPTR(W.)
170 W.=VARPTR(ARG0.):POKE PARAM.  ,PEEK(WP.):POKE PARAM.+1,PEEK(WP.+1)
180 CLS.=0:CLSS.=1:CLSH.=2:SCREEN ,3:DUMY=USR(CLS.):SCREEN 0,0:WIDTH 4
0,25
190 '
200 '  usr(0) check
210  FOR I=0 TO 4
220   LINE (I*60,I*20)-(640-I*60,168-I*20),7-I,BF
230  NEXT
240  LOCATE 0,23 : PRINT 'USR(CLS.)' : DUMY=USR(CLS.)
250  FOR DELAY=1 TO 1000:NEXT : LOCATE 0,23 : PRINT SPC(39)
260 '
270 '  usr(1) check
280  FOR I=0 TO 4
290   LINE (I*60,I*20)-(640-I*60,168-I*20),7-I,BF
300  NEXT
310  FOR BANK=0 TO 2
320   LOCATE 0,23 : PRINT USING 'USR(CLSS.) ; Bank=#';BANK
330   ARG0.=BANK : DUMY=USR(CLSS.) :FOR DELAY=1 TO 1000: NEXT
340   LOCATE 0,23 : PRINT SPC(39)
350  NEXT
360 '
370 '  usr(2) check
380  SCREEN 2,0
390  FOR I=1 TO 7
400   LINE (I*40,I*20)-(640-I*40,368-I*20),I MOD 2,BF
410  NEXT
420  LOCATE 0,23 : PRINT 'USR(CLSH.)' : DUMY=USR(CLSH.)
430  FOR DELAY=1 TO 1000 : NEXT : LOCATE 0,23 : PRINT SPC(39)
440 '
450 END
```

4.3.3 ROLL

グラフィック画面のスクロール命令としてROLL文がありますが，これはDisk-BASIC専用命令ですし，スクロールも上方向だけです．そこで，このライブラリでグラフィック画面をスクロールさせる機能を実現しました．

グラフィック画面をスクロールさせると言っても，スクロール・ライン数が決まっていれば実

に簡単なもので必要なライン分ずらしてブロック転送し，最後にスクロール分だけ消去すれば良いのです．

ところが，左右のスクロールでは話が変わってきます．グラフィック VRAM は1ドットを1 bit，つまり8ドットで1 byte ですが，これをスクロールさせるとなるとブロック転送命令は使えません．例えば，1 bit のスクロールでは，VRAM の値を1 bit ずつシフトしなければなりません．そこでプログラムを簡単にするため左右のスクロール幅は8ドット（1 byte）単位としました．この場合には，左右のスクロールもブロック転送によって済む訳で，例えば，左へ8ドットのスクロールは1 byte ずらして転送すれば良いのです．

この処理において面倒なのは，スクロール幅を可変としたため，ブロック転送の際の転送アドレスや転送バイト数をスクロール幅に応じて求めなければならない点です．

BASIC から呼ぶ時は，

```
ARG0.%=3:DUMY=USR(ROLUP.%)
```

とします（スクロール幅3ドットの場合）．上下方向のスクロールでは，スクロール幅は 1 ～199 の値を与えることができます．

また，左へ8ドットのスクロールでは，

図14　スクロールupの原理

図15　グラフィック・バンクの構成

図16 SCREEN 2 の時のスクロールupの原理

ARG 0 . %= 8 : DUMY=USR(ROLLE . %)

となりますが，左右スクロールの時には，このパラメータは８～632の値にしてください．左右のスクロール幅は８の倍数で指定しますが，８の倍数でない時，例えば９の時には切り上げて16と見なされます．

また，スクリーン・モードが２の時には，画面をバンク１と２を使って表示していますが，バンクの境界を越えてブロック転送命令を行えませんので，この時だけ交互にバンクを切り換えながら，バンク間で転送を行います．

USRの引数	パラメータ	機　能
3	スクロール幅を指定	スクロール UP (Screen 0)
4		スクロール UP (Screen 2)
5		スクロール DOWN (Screen 0)
6		スクロール DOWN (Screen 2)
7	スクロール幅を指定 (8ドット単位)	スクロール LEFT (Screen 0)
8		スクロール LEFT (Screen 2)
9		スクロール RIGHT (Screen 0)
10		スクロール RIGHT (Screen 2)

表7 ROLLのパラメータと機能

リスト11

```
100 '
110 ' graphic library ROLL test program
120 '
130 CLEAR ,&HE3FF : DEFINT A-Z : DIM F$(3)
140 DEF USR=&HE400 : PARAM.=&HE5F8
150 ARG0.=0
160 W.=0 : WP.=VARPTR(W.)
170 W.=VARPTR(ARG0.):POKE PARAM.  ,PEEK(WP.):POKE PARAM.+1,PEEK(WP.+1)
180 ROLUP.=3:ROLUPH.=4 : ' roll up
190 ROLDW.=5:ROLDWH.=6 : ' roll down
200 ROLLE.=7:ROLLEH.=8 : ' roll left
210 ROLRI.=9:ROLRIH.=10: ' roll right
220 FOR I=0 TO 3 : READ F$(I) : NEXT
230 DATA ROLUP,ROLDW,ROLLE,ROLRI
240 SCREEN ,3:DUMY=USR(CLS.):SCREEN 0,0:WIDTH 40,25
250 '
260 ARG0.=16
270  FOR I=0 TO 4
280   LINE (I*60,I*20)-(640-I*60,168-I*20),7-I,BF
290  NEXT
300 FOR FUNC=ROLUP. TO ROLRI. STEP 2
310  LOCATE 0,23
320  PRINT USING"USR(@.) scroll width=##";F$((FUNC-ROLUP.)/2),ARG0.
330  DUMY=USR(FUNC)
340  FOR DELAY=1 TO 1000:NEXT
350  LOCATE 0,23 : PRINT SPC(39)
360 NEXT
370 '
380 SCREEN 2,3:DUMY=USR(CLSH.):SCREEN 2,0 : ARG0.=32
390  FOR I=1 TO 7
400   LINE (I*40,I*20)-(640-I*40,368-I*20),I MOD 2,BF
410  NEXT
420 FOR FUNC=ROLUPH. TO ROLRIH. STEP 2
430  LOCATE 0,23
440  PRINT USING"USR(@H.) scroll width=##";F$((FUNC-ROLUPH.)/2),ARG0.
450  DUMY=USR(FUNC)
460  FOR DELAY=1 TO 1000:NEXT
470  LOCATE 0,23 : PRINT SPC(39)
480 NEXT
490 '
500 END
```

4.3.4 パターン反転

グラフィック画面上に作成したパターンを回転させたり，左右を反転させたりしたいと思うことがしばしばあります．このために作成したのが，ここで紹介するツールで，対称軸反転，x方向反転，y方向反転の3種類があります．

対称軸反転は，x軸とy軸を入れ替える変換で，左上と右下を結ぶ直線を軸として反転させます．x方向反転は左右の反転，y方向反転は上下の反転です．これらはすべて，グラフィック画面上にあるパターンに作用するものではなく，パターンをGETした配列内データに作用するものです．

では，この機能の使い方について説明しましょう．

まず，グラフィック画面上の反転させたいパターンをGET@文で配列に読み込みます．この時に読み込む領域は正方形（横方向と縦方向のドット数が同じ）になるようにしてください．次に反転されたパターンを書き込む配列を用意しますが，パターンの大きさは変化しませんから，配列は，反転前のパターンが入る大きさにしてください．そして，この2つの配列のアドレスをVARPTRで求めてパラメータとして持ってライブラリを呼び出すと，その配列に反転したパターンが，即使える形で入ってきます．後は，これをPUT@でグラフィック画面に表示させるだけです．

X 方向反転

Y 方向反転

対称軸反転

図17 パターン反転

GET@を実行した直後は，配列の最初の4byte には，2byte ずつ，x 軸と y 軸の GET した幅がドット数で入っています．そこで，ライブラリではこれより反転するパターンの大きさを調べていますが，この時にパターンが正方形であるものとしていますので，正方形でないときには，このパターンの実行結果は保障されません．GET されるパターンと GET された配列の内容の関係を図18に示しておきます．

この機能を呼び出す時の USR 関数の引数はそれぞれ表 8 の様になっています．

また，パラメータとして与える配列のアドレスは，配列要素の最初（通常添字 0 ）のアドレスとしてください．

GET@(0,0)−(7,7),P%

(0,0)

8 ドット

8 ドット

(7,7)

配列P%の内容

	配列P%の内容	
下位	08 H	…水平方向の幅
上位	00 H	
下位	08 H	…垂直方向の幅
上位	00 H	
→	00 H	…パターンのデータ
→	30 H	
→	70 H	
→	10 H	
→	10 H	
→	10 H	
→	10 H	
→	38 H	

図18 GETされるパターンとGETした配列の内容

USRの引数	パラメータ(ARG0.)	パラメータ(ARG1.)	機　　　能
11	GETした配列のアドレス	反転したパターンのための配列のアドレス	対称軸反転 (PROT)
12			X方向反転 (PROTX)
13			Y方向反転 (PROTY)

表8　パターン反転のパラメータと機能

この機能を使う時に，配列のアドレスを VARPTR で求めるのは，USR 関数でライブラリを呼ぶ直前に行ってください．なぜならば，BASIC は変数をプログラム中で出現した順番で登録しますが，配列は変列領域の常に直後から置くため，配列を宣言した後，まだ使っていない変数が現れると配列のアドレスがずれるからです．

リスト12

```
100 '
110 ' graphic library PROT test program
120 '
130 CLEAR ,&HE3FF : DEFINT A-Z
140 DEF USR=&HE400 : PARAM.=&HE5F8
150 ARG0.=0 : ARG1.=0 : ARG2.=0 : ARG3.=0
160 W.=0 : WP.=VARPTR(W.)
170 W.=VARPTR(ARG0.):POKE PARAM.  ,PEEK(WP.):POKE PARAM.+1,PEEK(WP.+1)
180 W.=VARPTR(ARG1.):POKE PARAM.+2,PEEK(WP.):POKE PARAM.+3,PEEK(WP.+1)
190 '
200 CLS.=0:CLSS.=1:CLSH.=2 : ' cls
210 PROT.=11:PROTX.=12:PROTY.=13 : ' pattern rotate
220 '
230 DIM P1(1601),P2(1601)
240 SCREEN 1,3,0,1 : ARG0.=0 : DUMY=USR(CLS.) : WIDTH 80,25 : SCREEN ,
0
250 CIRCLE (80,80),40,7
260 FOR I=0 TO 64 STEP 4
270  LINE(I,159)-(159,I),7 : LINE(159-I,159)-(0,159-I),7
280 NEXT
290 '
300 GET@(0,0)-(159,159),P1 : ARG0.=VARPTR(P1(0)):ARG1.=VARPTR(P2(0))
310 LOCATE 6,20
320 PRINT"original                PROT.                  PROTX.
 PROTY."
330 DUMY=USR(PROT.) : PUT@(160,0),P2,PSET
340 DUMY=USR(PROTX.): PUT@(320,0),P2,PSET
350 DUMY=USR(PROTY.): PUT@(480,0),P2,PSET
360 END
```

この写真は640×200の時の実行例ですが，640×400ですと，きれいな正方形となります．

4.3.5 クロス・カーソル

グラフィック画面上で，ドット単位で位置を求めるのは結構大変です．この時のために，グラフィック上でのカーソル機能を作りました．USR(14)と USR(15)がこれに当ります．USR(14)がカーソルの表示及び消去，USR(15)がカーソルの移動を任っています．

USR(14)は，カーソルを表示する点と，カーソルの表示範囲とをパラメータに与えて呼ぶと，グラフィック上にクロス・カーソルが表示されます．ここで表示範囲を指定して可変にしているのは，画面上すべてをカーソルの表示エリアとするとテキスト画面の文字とカーソルが重なって見づらくなるのを避けるためで，ここで指定した位置と画面の左上の点（0，0）からなる長方形内だけに表示されます．

また，このUSR(14)はトグル動作になっており，カーソルが表示されている時に実行するとカーソルが消去され，消去されている時には表示されます．

カーソルを動かすのはUSR(15)を使いますが，ライブラリの内で，この機能だけがUSR関数の返す値の中で唯一意味のある値となっています．

USR(15)は，呼ばれるごとにテン・キーをスキャンして，押されているキーの方向へ動きます．

(0, 0)
カーソル表示範囲
USR(14)のパラメータで指定した点
(639, 199)

図19　クロス・カーソルの表示と移動

USRの引数	パラメータ				機能
	ARG 0.	ARG 1.	ARG 2.	ARG 3.	
14	カーソルの水平位置	カーソルの垂直位置	カーソル最大値（水平位置）	カーソル最大値（垂直位置）	カーソル表示及び表示範囲設定．表示されている時はカーソル消去．
15					カーソル移動．

表9　クロス・カーソルのパラメータと機能

またこの時，シフト・キーも同時にスキャンしており，シフト・キーも押されていると，カーソルは8ステップずつ移動します．その結果，カーソルが移動した時には255，位置が変わっていない時には0をそれぞれ返します．また，この機能にもパラメータ用として変数が2つあり，これには現在のカーソル位置が代入されて返されます．

このカーソルの表示は，バンク1にXORで書き込んでいますので，カーソルの移動によって

グラフィック画面上のパターンが消えることはありません．さらに，バンク1だけに書き込みを行っていますから，カーソルの色は常に青です．また，このカーソル機能は SCREEN　0 のカラー・モード用ですが，白黒モードでも実行は可能です．但し，色が2色のため表示が多少見苦しくなってしまいます．

リスト13

```
100 '
110 ' graphic library GCREV & GCMOVE test program
120 '
130 CLEAR ,&HE3FF : DEFINT A-Z
140 DEF USR=&HE400 : PARAM.=&HE5F8
150 ARG2.=639 : ARG3.=191 : ARG0.=ARG2./2 : ARG1.=ARG3./2
160 W.=0 : WP.=VARPTR(W.)
170 W.=VARPTR(ARG0.):POKE PARAM.  ,PEEK(WP.):POKE PARAM.+1,PEEK(WP.+1)
180 W.=VARPTR(ARG1.):POKE PARAM.+2,PEEK(WP.):POKE PARAM.+3,PEEK(WP.+1)
190 W.=VARPTR(ARG2.):POKE PARAM.+4,PEEK(WP.):POKE PARAM.+5,PEEK(WP.+1)
200 W.=VARPTR(ARG3.):POKE PARAM.+6,PEEK(WP.):POKE PARAM.+7,PEEK(WP.+1)
210 CLS.=0:CLSS.=1:CLSH.=2 : ' cls
220 GCREV.=14:GCMOVE.=15 : ' G-cursor
230 '
240 ON STOP GOSUB *BREAK : STOP ON
250 WIDTH 40,25 : SCREEN 0,3 : DUMY=USR(CLS.) : SCREEN ,0
260 DUMY=USR(GCREV.) : COL=4
270 *LOOP
280  FLAG=USR(GCMOVE.)
290  PSET (ARG0.,ARG1.),COL
300  LOCATE 0,24,0
310  PRINT USING "ARG0.=###  ARG1.=###  USR(GCMOVE.)=###";ARG0.,ARG1.,
FLAG;
320  GOTO *LOOP
330 '
340 *BREAK
350  STOP OFF : LOCATE ,,1
360 END
```

4.3.6　漢字1行PUT

このサブルーチンは5章で紹介する漢字ワード・プロセッサのために作成したもので，詳しい使い方はそちらのプログラムを読んでください．

機能としては，グラフィック画面に漢字を横40文字，縦は20行，または10行で表示する事として，指定した行に1行分表示します．1行分の漢字コードは，次の図の様な文字で与えられますが，このライブラリを呼ぶ時には，文字列が代入されているストリング変数のアドレスをパラメータとして渡します．

ここでサポートしているのは，漢字ワード・プロセッサで使う，16×16ドットの漢字，記号及び英数字だけで，半角文字や1/4角文字は考慮していません．

USR(16)と USR(17)が，漢字1行 PUT となっており，USR(17)は SCREEN　2（縦20行）の時のためのもので，表示行は0～19の値で与えます．

USR(16)は，SCREEN 1の白黒3ページ・モード用であり，この場合は，さらに書き込むバンクもパラメータとして与えます．

ではここで，漢字ROMのI／Oポートについて説明しておきましょう．

漢字パターンは128KbyteのROMに格納されており，これはI／Oポートを介してCPUに接続されています．この漢字出力のために使われているポートを次の表に示します．まず，漢字ROMへ与えるアドレスはE9HとE8Hにそれぞれ上位と下位アドレスを書き込み，EAHに何

ストリング変数
変数名
長さ
ストリング・バッファ
漢字1文字

図20 漢字1行PUTパラメータ

USRの引数	パラメータ			機能
	ARG 0.	ARG 1.	ARG 2.	
16	文字変数のアドレス	表示行	漢字表示バンク	指定されたバンクの指定行に1行分PUT
17				指定行に1行PUT

表10 漢字1行PUTのパラメータと機能

か書いた後に（値は意味を持たない），E9H（上位），E8H（下位）を読むと，漢字ROMからの出力が得られます．そして，EBHになんらかの値を出力する事で，ROMの読み出しを停止します．

ここで注意しなければならないことは，ROMに与えるアドレスはJIS漢字コードではないことです．漢字パターンは，16×16ドットで，1文字あたり32byte（4bit）必要なので，アドレスのうち残り12bitで漢字を選択しなければならないのに，漢字コードは16bitで指定するため，漢字コードを直接ポートに書き込むことはできません．そこで，ROMにはJISコードのアドレスのビット位置を変換して与えることになります．また，1文字に必要な漢字パターンのデータは，16×16ドット文字で32byte，半角文字で16byte，¼角文字で8byteですが，それぞれ漢字ROMに与えるアドレスの下位4bit，3bit，2bitがパターンのRAWアドレス（列ならび）となっています．

それを次の表に示します．

1/4角文字	漢字コード (2121H～2771H)
C_{13} ～ C_0 ⇒ A_{15} ～ A_2	C_6, C_5 → A_{13}, A_{12} 0 → A_{15}, A_{14} C_{10} ～ C_8 → A_{11} ～ A_9 C_4 ～ C_0 → A_8 ～ A_4
半 角 文 字	**漢字コード (3021H～4F53H)**
C_{12} ～ C_0 ⇒ A_{15} ～ A_3	C_6, C_5 → A_{15}, A_{14} C_{12} ～ C_8 → A_{13} ～ A_9 C_4 ～ C_0 → A_8 ～ A_4

C_{15}～C_0：JIS漢字コード
C_{15}～A_0：漢字ROMアドレス
表11　漢字コード・漢字ROMアドレス変換表

リスト14

```
100 '
110 ' graphic library KPUT & KPUTH test program
120 '
130 CLEAR ,&HE3FF : DEFINT A-Z
140 DEF USR=&HE400 : PARAM.=&HE5F8
150 ARG0.=0 : ARG1.=0 : ARG2.=0
160 W.=0 : WP.=VARPTR(W.)
170 W.=VARPTR(ARG0.):POKE PARAM.  ,PEEK(WP.):POKE PARAM.+1,PEEK(WP.+1)
180 W.=VARPTR(ARG1.):POKE PARAM.+2,PEEK(WP.):POKE PARAM.+3,PEEK(WP.+1)
190 W.=VARPTR(ARG2.):POKE PARAM.+4,PEEK(WP.):POKE PARAM.+5,PEEK(WP.+1)
200 '
210 CLS.=0:CLSS.=1:CLSH.=2 : ' cls
220 KPUT.=16:KPUTH.=17 : ' kanji put
230 '
240 WIDTH 40,25 : SCREEN ,3 : DUMY=USR(CLS.) : SCREEN 1,0,0,1
250 LOCATE 0,15 : PRINT "ｺﾉ Program ﾆﾊ ｶﾝｼﾞ ROM ｶﾞ ﾋﾂﾖｳ ﾃﾞｽ｡"
260 PRINT : PRINT "ﾏﾀ ｺｳﾌﾞﾝｶｲﾉｳ Graphic display ｶﾞ ﾅｲﾄ"
270 PRINT "KPUTH. ﾉ check ﾊ ﾃﾞｷﾏｾﾝ｡"
280 FOR DELAY=1 TO 4000:NEXT
290 FOR BANK=0 TO 2
300  FOR KLINE=0 TO 9
310   LOCATE 0,24
320   PRINT USING"KPUT. check   bank=#  line=##";BANK,KLINE;
330   SCREEN ,,,2^BANK
340   K$="  " : FOR K=1 TO 40 : K$=K$+MKI$(&H3021+BANK*10+KLINE) : NEX
T
350   ARG0.=VARPTR(K$) : ARG1.=KLINE : ARG2.=BANK : DUMY=USR(KPUT.)
360  NEXT
370 NEXT
380 SCREEN ,,,7
390 FOR DELAY=1 TO 3000:NEXT
400 '
410 SCREEN 2,3 : DUMY=USR(CLSH.) : SCREEN ,0
420 LOCATE 0,24 : PRINT SPC(39);
430 FOR KLINE=0 TO 19
440  LOCATE 0,24 : PRINT USING"KPUTH. check   line=##";KLINE;
450  K$="  " : FOR K=1 TO 40 : K$=K$+MKI$(&H2421+KLINE) : NEXT
460  ARG0.=VARPTR(K$) : ARG1.=KLINE : DUMY=USR(KPUTH.)
470 NEXT
480 END
```

640×400の時の実行例

4.3.7 入力の仕方

このライブラリは，すべてアセンブラで書かれており，約1.5Kbyteの大きさがあります．このライブラリにはそれなりの機能がありますし，また本書はこのライブラリがないと動かないプログラムがありますから，(Gエディタ，漢字ワード・プロセッサ，logic，アニメーション・ジェネレータ）できるだけ入力するようにしてください．

これを入力する前には必ず，

```
CLEAR , &HDFFF
```

を実行しておきます．そして，MONコマンドでモニタに入り，SまたはEコマンドでダンプ・リスト(リスト15)を見ながら，アドレスE000HからE5FFHまでのデータを打ち込みます．かなり量があり，BASICプログラムとは違いますから，コーヒーでも飲みながら落ち着いて入力すればよいでしょう．

ひと通り打ち終ったら，まずSAVEをしますが，対象がカセットかディスクかで方法が異なります．

カセットの場合には，モニタのWコマンドで，

```
h] W, E000, E5FF
```

ディスクの場合にはBASICに戻り，

```
BSAVE  ファイル名, &HE000, &H600
```

を実行してください．

これはアドレスです．データではありません．

リスト15

```
E000 : F3 21 18 E0 11 00 80 01  90 03 DB 70 F5 3E 80 D3
E010 : 70 ED B0 F1 D3 70 FB C9  26 00 79 D6 0A 38 02 4F
E020 : 24 7C C6 5C C5 4F EB 7C  FE 30 7D 30 05 1F E6 30
E030 : 18 03 17 E6 C0 47 7D 17  17 17 E6 F8 17 6F 7C 17
E040 : E6 3F B0 67 22 77 83 E1  CB 24 5C 7D 17 85 17 C6
E050 : C0 57 AF CB 3D 1F CB 3D  1F 65 6F 19 06 10 ED 79
E060 : EB 2A 77 83 7C D3 E9 7D  D3 E8 23 22 77 83 AF D3
E070 : EA DB E9 67 DB E8 6F AF  D3 EB EB 72 23 73 11 4F
E080 : 00 19 10 DC D3 5F C9 3E  02 CD 7A 80 3E 01 CD 7A
E090 : 80 AF C6 5C 4F ED 79 01  7F 3E 11 01 C0 21 00 C0
```

```
E0A0 : 36 00 ED B0 D3 5F C9 CD  5B 81 21 00 C0 19 22 79
E0B0 : 83 3E 5C CD CD 80 3C FE  5F 20 F8 C9 CD 5B 81 21
E0C0 : 00 C0 19 22 79 83 3E 5C  CD E2 80 21 00 C0 ED 4B
E0D0 : 81 83 D3 5D 7E EB D3 5C  77 EB 23 13 0B 78 B1 20
E0E0 : F1 D3 5F 3E 5D CD E2 80  62 6B 13 4F ED 79 36 00
E0F0 : ED 4B 81 83 0B ED B0 D3  5F C9 2A 79 83 11 00 C0

E100 : 4F ED 79 ED 4B 7D 83 ED  B0 D3 5F C9 CD 5B 81 11
E110 : 00 C0 19 2B 22 79 83 3E  5E CD 34 81 3D FE 5B 20
E120 : F8 C9 CD 5B 81 11 00 C0  19 2B 22 79 83 3E 5D CD
E130 : 49 81 21 7F FE ED 4B 81  83 D3 5C 7E EB D3 5D 77
E140 : EB 2B 1B 0B 78 B1 20 F1  D3 5F 3E 5C CD 49 81 62
E150 : 6B 1B 4F ED 79 36 00 ED  4B 81 83 0B ED B8 D3 5F
E160 : C9 2A 79 83 11 7F FE 4F  ED 79 ED 4B 7D 83 ED B8
E170 : D3 5F C9 A7 28 1D FE C8  30 19 26 00 6F 54 5F 29
E180 : 29 19 29 29 29 29 22 81  83 EB 21 80 3E ED 52 22
E190 : 7D 83 C9 33 33 C9 CD 13  82 3E 5E 18 05 CD 13 82
E1A0 : 3E 5D 21 00 C0 19 22 79  83 21 50 C0 ED 52 22 7B
E1B0 : 83 2A 79 83 11 00 C0 4F  ED 79 ED 4B 7D 83 ED B0
E1C0 : 32 83 83 2A 7F 83 EB 2A  7B 83 0E C8 3A 81 83 47
E1D0 : 36 00 23 10 FB 19 0D 20  F6 3A 83 83 3D FE 5B 20
E1E0 : D0 D3 5F C9 CD 13 82 3E  5E 18 05 CD 13 82 3E 5D
E1F0 : 21 7F FE ED 52 22 79 83  2A 79 83 11 7F FE 4F ED

E200 : 79 ED 4B 7D 83 ED B8 32  83 83 2A 7F 83 EB 21 00
E210 : C0 0E C8 3A 81 83 47 36  00 23 10 FB 19 0D 20 F6
E220 : 3A 83 83 3D FE 5B 20 D0  D3 5F C9 21 07 00 19 7D
E230 : CB 3C 1F CB 3C 1F CB 3C  1F 26 00 6F 22 81 83 EB
E240 : 21 50 00 A7 ED 52 22 7F  83 21 80 3E ED 52 22 7D
E250 : 83 C9 ED 53 85 83 ED 43  87 83 DD 22 89 83 FD 22
E260 : 8B 83 CD 34 83 C3 F8 82  AF 32 8F 83 21 8D 83 36
E270 : 08 DB 08 E6 40 28 02 36  01 DB 00 2F 57 DB 01 2F
E280 : 5F 01 00 00 E6 02 20 05  7A E6 48 28 01 04 7A E6
E290 : 92 28 01 05 7A E6 0E 28  01 0C 7A E6 80 20 05 7B
E2A0 : E6 03 28 01 0D 78 B1 C4  9E 82 ED 5B 85 83 ED 4B
E2B0 : 87 83 3A 8F 83 C9 3A 8F  83 F6 FF 32 8F 83 78 2A
E2C0 : 8D 83 EB 21 00 00 A7 28  15 FA B5 82 EB ED 52 EB
E2D0 : 2A 85 83 19 EB 2A 89 83  A7 ED 52 EB 30 03 2A 85
E2E0 : 83 C5 E5 CD 34 83 E1 22  85 83 CD 34 83 C1 3A 8B
E2F0 : 83 3C 47 79 A7 3A 8D 83  4F 3A 87 83 28 0A FA ED

E300 : 82 81 B8 38 03 91 38 F9  F5 CD F8 82 F1 32 87 83
E310 : 2A 87 83 29 29 29 29 54  5D 29 29 19 11 00 C0 19
E320 : E5 ED 4B 89 83 79 CB 38  CB 19 CB 38 CB 19 CB 38
E330 : CB 19 E6 07 16 00 5F 21  67 83 19 41 4E E1 D3 5C
E340 : 7E 2F 77 23 10 FA 79 AE  77 D3 5F C9 2A 85 83 7D
E350 : CB 3C CB 1D CB 3C CB 1D  CB 3C CB 1D 11 00 C0 19
E360 : EB 21 6F 83 E6 07 06 00  4F 09 4E EB 11 50 00 3A
E370 : 8B 83 3C 47 D3 5C 79 AE  77 19 10 FA D3 5F C9 80
E380 : C0 E0 F0 F8 FC FE FF 80  40 20 10 08 04 02 01 00
E390 : 00 00 00 00 00 00 00 00  00 00 00 00 00 40 01 63
E3A0 : 00 7F 02 C7 00 01 00 00  74 04 4F CD FB 26 CD E9
E3B0 : 0A CD DC 0A FE 29 C4 74  04 C3 F3 0A 3E 01 32 AB
E3C0 : 37 C3 F3 0A AF 32 AB 37  C3 F3 0A AF 32 A9 37 C3
E3D0 : F3 0A 3E 01 32 A9 37 C3  F3 0A 3E FF 32 AA 37 C3
E3E0 : F3 0A AF 32 AA 37 C3 F3  0A 3E 01 32 AA 37 C3 F3
E3F0 : 0A 3E 01 32 69 37 C3 F3  0A AF 32 69 37 C3 F3 0A

E400 : F3 FE 02 20 2F 22 E1 E5  AF 32 E3 E5 23 7E A7 20
E410 : 23 57 2B 7E FE 13 30 1C  87 87 5F 21 52 E4 19 5E
E420 : 23 56 23 4E 23 46 C5 CD  46 E4 CD 50 E4 E1 CD 51
E430 : E4 CD 40 E4 2A E1 E5 3A  E3 E5 77 23 36 00 FB C9
E440 : 3A 3D E5 D3 70 C9 DB 70  32 3D E5 3E 80 D3 70 C9
E450 : EB E9 45 E4 6F 80 9A E4  7A 80 45 E4 74 80 9A E4
E460 : 8F 80 9A E4 A4 80 9A E4  F4 80 9A E4 0A 81 BD E4
E470 : 7E 81 BD E4 85 81 BD E4  CC 81 BD E4 D3 81 B7 E4
E480 : 3E E5 B7 E4 7C E5 B7 E4  9B E5 9F E4 3A 82 50 82
E490 : 2D E5 45 E4 C4 E4 45 E4  CD E4 2A F8 E5 7E C9 2A
E4A0 : FC E5 CD C0 E4 DD 21 00  00 DD 19 2A FE E5 CD C0
E4B0 : E4 FD 21 00 00 FD 19 2A  FA E5 4E 23 46 2A F8 E5
E4C0 : 5E 23 56 C9 2A FC E5 7E  21 0A 80 18 03 21 00 80
E4D0 : 32 DA E5 22 DD E5 2A FA  E5 6E 26 FF 22 DB E5 2A
E4E0 : F8 E5 5E 23 56 EB 7E 23  D6 02 28 24 5E 23 56 EB
```

```
E4F0 : 23 23 1F 47 C5 ED 4B DB  E5 04 ED 43 DB E5 5E 23

E500 : 56 23 3A DA E5 E5 2A DD  E5 CD 51 E4 E1 C1 10 E4
E510 : ED 4B DB E5 2A DD E5 04  78 FE 28 30 0F 11 21 21
E520 : 3A DA E5 C5 E5 CD 51 E4  E1 C1 18 EB C9 2A F8 E5
E530 : 73 23 72 2A FA E5 71 23  70 32 E3 E5 C9 00 CD B6
E540 : E5 FD 2A DF E5 47 5F C5  2E 80 FD 22 DF E5 4B 06
E550 : 08 DD 66 00 DD 23 7D CB  24 30 05 FD B6 00 18 04
E560 : 2F FD A6 00 FD 77 00 FD  19 10 EB 0D 20 E1 FD 2A
E570 : DF E5 CB 3D 30 D4 FD 23  C1 10 CC C9 CD B6 E5 4B
E580 : 5F 19 43 DD 7E 00 DD 23  C5 06 08 4F CB 21 1F 10
E590 : FB C1 2B 77 10 ED 0D C8  19 18 E6 CD B6 E5 47 19
E5A0 : 10 FD 4B 5F ED 52 43 DD  7E 00 DD 23 77 23 10 F7
E5B0 : 0D C8 ED 52 18 EE D5 DD  E1 AF 60 69 DD 4E 00 71
E5C0 : 23 77 23 DD 46 02 70 23  77 23 22 DF E5 11 04 00
E5D0 : DD 19 58 78 0F 0F 0F E6  1F C9 00 00 00 0A 80 00
E5E0 : 00 00 00 00 EB 2A 59 38  EB 73 23 72 E1 C3 5D 19
E5F0 : F5 CD E9 44 3A C9 39 B7  C4 29 2B 3A 71 37 B7 C4
```

上記の手続きが終了したら，打ち込んだプログラムのチェックをします．次のプログラムは，このライブラリのためのチェック・プログラムです．このプログラムを実行してチェックを行ってください．但し，このプログラムだけは，絶対にタイプミスをしてはいけません．

リスト16

```
100 '
110 ' Graphic library SUM CHECK program
120 '
130 CLEAR ,&HDFFF : DEFINT A-Z
140 DEF FNH$(H)=RIGHT$("000"+HEX$(H),4)
150 ADR=&HE000 : FAIL=0
160 FOR AH=0 TO &H5C0 STEP 64
170  READ SUM$ : SUM=VAL("&h"+SUM$)
180  FOR AL=0 TO 63
190   SUM=SUM-PEEK(ADR+AH+AL)
200  NEXT
210  IF SUM=0 THEN *SKIP
220   PRINT FNH$(ADR+AH)" ｶﾗ "FNH$(ADR+AH+63)" ﾏﾃﾞ ﾉ ｱｲﾀﾞﾃﾞ ﾆｭｳﾘｮｸ ﾐｽ
ｶﾞ ｱﾘﾏｽ｡"
230   FAIL=FAIL+1
240  *SKIP
250 NEXT
260 PRINT
270 IF FAIL=0 THEN PRINT "complete !"
280 END
290 '
300 DATA 1C39,213E,1C8B,1E1F,1F1D,1E40,1970,1B36
310 DATA 1A18,1C37,197F,1F47,1B7B,1B69,14D4,1CC2
320 DATA 1F60,2841,263F,1E11,2468,206D,1D58,175F
```

実行すると，まず64byte単位でサム・チェックを行い，完全に合っていると最後に〝complete！〟と表示しますので，それまで何回も実行し，再度違うファイル名でSAVEしておいてください．以後このライブラリは必要に応じてロードして使います．

4.3.8 ライブラリのロード

このライブラリは，メモリ上に常駐して使いますので，使う前に次の操作を1度だけ行っておいてください．

a) Disk-BASICの場合

次のプログラムを実行してください．

リスト17

```
100 '
110 ' Graphic libray initialize program (DISK)
120 '
130 CLEAR ,&HDFFF : DEFINT A-Z
140 DEF USR=&HE000 : ADR=&HE000
150 INPUT "BSAVE ｼﾀ library ﾉ FILE NAME ﾊ";F$
160 BLOAD F$,ADR
170 DUMY=USR(0)
180 CLEAR ,&HE3FF
190 END
```

b) ROM-BASICの場合

まず,

CLEAR , &HDFFF

を実行してから MON でモニタに入り，モニタ上の〝R〟コマンドで，カセットからライブラリの機械語プログラムをロードします．

そして，次のプログラムを実行させてください．

リスト18

```
100 '
110 ' Graphic libray initialize program (ｶｾｯﾄ)
120 '
130 DEF USR=&HE000
140 DUMY=USR(0)
150 CLEAR ,&HE3FF
160 END
```

これでライブラリのロードは終りですが，この時に走らせたプログラムは E000H から E017H の転送プログラムです．この節の画面クリアの所で，このライブラリは E400H から E5FFH と，メイン・バンクの8000H から83FFH の2つの部分になっていると説明しましたが，この転送プログラムが呼ばれた時に E018H から E3FFH までの内容がテキスト・ウィンドウを介して転送されているのです．

4.3.9 その他のグラフィックI／O

さて，このライブラリでは，いくつかのグラフィック関係のI／Oを使いましたが，ここでライブラリでは使わなかったポートを紹介しましょう．

a) CRTC

テキストV-RAM は CRTC（μPD-3301）を使って表示していますが（第1章参照），CRTC は50H がパラメータ・レジスタ，51H がコマンド・レジスタとなっています．CRTC の機能については CRTC のマニュアルを御覧ください．

b) ボーダ・カラー，バック・グラウンド・カラー　コントロール・ポート

COLOR 文の3番目のパラメータはボーダー・カラーの指定で，これによりグラフィックが表示されるエリア外の色を付けますが，高解像度ディスプレイを使用している時には BASIC では

黒以外許されていません．これはボーダ・カラーを黒以外の色にすると，その内側の色が影響を受けて，異なった色調になってしまうからですが，これを逆に利用しポートに直接データを書き込みボーダー・カラーを黒以外にするとおもしろい色調が表現できます．

また，白黒モード（SCREEN 1，SCREEN 2）の時のバック・グラウンド・カラーも，同様にポートから直接変える事ができます．

このどちらも，52H のポートに書き込みますが，bit6 から 4 がバック・グラウンド・カラー，bit2から 0 がボーダー・カラーと設定されています．

╳	緑	赤	青	╳	緑	赤	青
	バック・グラウンド・カラー				ボーダー・カラー		

表12　バック・グラウンド，ボーダー・カラー　セレクト・ポート

c) グラフィック・テキスト　ディスプレイ・コントロール・ポート

SCREEN 1の時には，グラフィックは白黒 3 ページとなりますが，このモードでは表示するページ，すなわちバンクを色々に指定し，画面を合成したりできます．ポート53H の bit 1から 3 が，このための bit で，0 の時には表示，1 の時には表示禁止となります．また，このポートの bit0によりテキスト VRAM の表示も同様に制御出来ます．

D7–D4	D3			D0	
╳					53H
	バンク 3	バンク 2	バンク 1	テキストVRAM	

1の時　表示　禁止
0の時　表示．

表13　ディスプレイ　コントロール・ポート

d) カラー・パレット　コントロール・ポート

N_{88}-BASIC では，カラー・パレットを変える事でグラフィック上の図形の色を瞬時に変える事ができますが，これに使われている I／Oポートが，54H から5BH にあります．このポートへの直接書き込みによっても，カラー・パレットを変える事が出来ます．

	D7〜D4	D3			D0	
パレット0	×	×				54H
	×	×				55H
	×	×				56H
	×	×				57H
	×	×				58H
	×	×				59H
	×	×				5AH
パレット7	×	×				5BH
			緑	赤	青	

表14　カラーパレット　セレクトポート

5章 ビジネスへの応用

コンピュータ・グラフィックスの特徴の1つに，様々な情報を視覚的に表現でにるということがあります．本章では，この特徴を活かしてビジネス・グラフの作成をしてみます．また，最近ビジネスの分野で，特に注目されている漢字ワード・プロセッサの完全リストもここで紹介します．これは一見グラフィックスとは無関係のように思われますが，PC-8801の漢字出力はグラフィック画面に行われますから，少からずグラフィックスの知識を必要とします．

5.1 グラフを描く

ここではビジネス・グラフの中でグラフィックスの要素を含むものとして折れ線グラフ，円グラフ，レーダー・チャート，シンボル・グラフ，対数グラフを例にとり，グラフ作成のためのプログラムについて説明していきます．但し，データの入力及びその保存などについては，本書の目的とするところではありませんので，それについては詳しく触れません．

5.1.1 ウィンドウとビューポートの活用

N_{88}-BASICのウィンドウ機能とビューポート機能は，グラフを作成するうえで，非常に有益です．まず，グラフの有効表示領域（画面全体の内，点や線を描く領域）をビューポートとして指定し，さらに，その領域にグラフのスケールに合わせてウィンドウを指定します．そうすれば，データの値をグラフ上の座標に変換するのにほとんど計算は不要になります．

このウィンドウとビューポートについては，3章の〝任意の関数をプロットする〞で詳しく説明しましたので，もうお分かりでしょう．但し，y軸の正負が反転していることを忘れないで，正しく補正を行わなければなりません．

では，グラフのスケールの決め方から説明していきましょう．

5.1.2 スケールの決め方

データをグラフ上に表す場合，グラフの目盛について良く考えなければなりません．目盛の間

隔を決めるときは，以下のような注意が必要です．

○はみだしてしまうデータがないこと．
○目盛の最小値，最大値が切りのいい値であること．
○目盛の幅が切りのいい値であること．

これは，人間がデータを眺めて決定しても良いのですが，せっかくコンピュータを使うのですから，ついでに処理します．

まず，データの中から最大値を探します．最小値は大抵のグラフでは0ですが，対数グラフやデータが一部分に集中している場合などでは求めた方がよいでしょう．そして，この最大値より大きく，なおかつ切りのいい値をグラフの最大値として決定します．問題は，切りのいい値をどのように決定するかということです．これはグラフの種類にもよりますが，ここでは $a \times 10^n$（aは1～9の自然数，nは整数）としました．この値を求めるには，各種数値関数を用いて求める方法と，数値を文字列に置き換えて求める方法がありますが，ここでは数値関数で求める方法を用いました．これは次の様な方法で求めることができます．

〈グラフ最大値〉＝(INT(〈データ最大値〉／10∧〈桁数〉)＋1)＊10〈桁数〉
但し，
〈桁数〉＝INT（LOG（〈データ最大値〉)／LOG（10))

次に，目盛の幅について考えてみましょう．x軸の方は，多くの場合データが等間隔で与えられるので(年，月など)，その値で振れば問題ありません．等間隔でない場合は，y軸と目盛の幅と同じ様に決定しなければなりません．

では，y軸の目盛の幅について考えてみましょう．まず，一番簡単な方法としては，グラフの最大目盛の値にかかわらず，10等分とか5等分してしまうことです．これは，前に説明した目盛の幅が切りのいい値である．ということに反するように思うかもしれませんが，最大目盛を切りのいい値にしてあるため，さほど問題はなく十分実用になります．

次に考えつくのは，最大目盛が $a \times 10^n$ で表されていることから，最大目盛をa等分するという方法です．このとき，aが1なら10等分とします．こうすると，各目盛は切りのいい値になります．しかし，aが2や3のとき目盛の間隔が広くなりすぎ，データの値がどれくらいなのか分かりにくくなってしまいます．そこで，aの値に応じて，何等分にするかを変えるようにします．例えば，aが5以上のときはa等分，aが4以下のときは（2a）等分という様にです．こうすれば，そのままa等分するよりはだいぶ見易くなります．

5.1.3 数字の振り方

いくら目盛をきちんと定めても，それがどんな値を示すかが分からなければ，グラフとして役

に立ちません．そこで，グラフの軸に数字を振ることについて考えてみましょう．

すでに，目盛の最大値や幅は決定されているので，後は，目盛の位置に合うようにキャラクタ座標を計算するだけです．ただ，キャラクタ座標系は最大でも，80×25とグラフィック座標に比べて荒く，また，（0，0）—（79，24）の絶対座標しかないため，少し計算は面倒になります．

キャラクタ座標の計算は，丁度オリジナル・スクリーン座標系のみを使用したときの位置決定と同じです．x軸を例にとって説明しましょう．

まず，図1のようにグラフの原点に対応するキャラクタ座標を(x_o, y_o)，最大目盛に対応する座標を(x_m, y_o)とします．そして，この間をa等分して目盛を振るとすると，n番目の目盛に振られる数字のキャラクタ座標のxの値は，次の様になります．但し，nは0から始まるものとします．

$$x = x_o + INT((x_m - x_o) * n/a + 0.5) - l$$

l（エル）というのは，数字の桁数です．1を引くことにより，数字を右詰めで出力したときに，目盛の位置と数字の右端が合う様になります．また，0.5を加えて整数化するのは，四捨五入するためです．

y軸

x軸

0

(x_o, y_o)

最大目盛

(x_m, y_o)

図1

y軸の場合は，キャラクタ座標系もグラフィック座標と同じ様に上方が0のため，引き算をして正しい値にしなければなりません．

数字を出力する座標は求められましたが，まだ1つ問題があります．それは目盛が細かく，出力する数字が多いときです．x軸の場合，数字が多かったり，あるいは桁数が多かったりすると，数字間のスペースが無くなったり，重なったりして非常に見苦しくなります．そこで，1目盛おき，2目盛おきに数字を振るようにプログラムを工夫しなければなりません．キャラクタは横80字ですから，どんなに詰めてもx軸については20組位しか数字は振れません．見易くするためには，10組程度にすべきでしょう．この章で取り上げている折れ線グラフは，x軸は最大21組の目盛を振りますが，目盛が11以上のときは，1つ置きに数字を振る様にしています．

y軸の方は，行数の制限から20組程度しか振れませんが，見易さの点から，やはり10組程度に押さえた方が良いでしょう．

ここで，キャラクタ画面とビューポートについて注意することがあります．それは，ビューポ

ートを設定する際，右端をどこまで持ってくるかということです．実際には639，つまり画面の一番右端まで設定できる訳ですが，こうすると数字を振るときに不都合が生じることがあります．それは，1行の80文字目に文字を出力した後は，たとえ，セミコロン（；）を付けてある PRINT 文であっても改行してしまうからです．特に，それが25行目であった場合，キャラクタ画面全体がスクロール・アップしてしまい，せっかく計算した数字が全部ずれてしまいます．それを妨ぐために，右端1～2文字分は余裕をもってビューポートを設定しなければなりません．

5.2 各種グラフ

これから，実際にグラフを描くプログラムについて説明していきます．ここに挙げているプログラムは，すべてサブルーチン・ライブラリとして作られており，データさえ正しく引き渡せば使えるようになっています．そのため，ラベル名や変数名は，ライブラリをコールするメイン・プログラムと重ならないように〝.XX″という形で振張子が付けられています（但し，説明の際は拡張子を省略します）．

また，これらのサブルーチンには，データやデータの数といった直接必要なパラメータ以外にも，いくつか引き渡さなければならないデータがあります．例えば，折れ線グラフのライン・スタイルや色，シンボル・グラフのシンボルのパターンなどです．これらは，各サブルーチンを呼び出す度に設定したのでは，時間もかかり，またデータ文のポインタの管理なども大変になってしまうため，データ設定のためのサブルーチンをメイン・プログラムから呼ぶようにします．このサブルーチンは，グラフ用のサブルーチンを呼ぶ前に一度だけ呼びます．メイン・プログラムが DATA 文を使用している時は，データ設定用のサブルーチンの DATA 文と，本体のデータ文がずれたりしない様に注意してください．また，配列の宣言もここで行うので，データの代入はこれを呼んだ後に行ってください．

☆画面モードについて

各サブルーチンは，640×200のカラー・モードを使っています．もし，640×400の白黒モードで使う場合は，VIEW 文やカラー指定などを書き換えなければなりません，また，シンボル・グラフはオリジナル・スクリーン座標を使っているため，さらに書き換える部分があるので注意してください．

ところで，これらのライブラリが必ずしもユーザー・プログラムの要求を満たすものとは限りません．あくまで，グラフを描くためのプログラムの参考例と考えてください．また，グラフを描くための補間，近似などの数値処理は，本書の主目的ではないので行っていません．

5.2.1 折れ線グラフ

折れ線グラフは，最もポピュラーなグラフでしょう．これを描くプログラムをリスト1に示し

ます．100行台が呼び出しのサンプル・プログラム，40000行台がデータ設定プログラムで，線の色，ライン・スタイルを設定しています．50000行台がグラフを描くサブルーチンです．このサブルーチンに使われる変数，ラベル名はすべて〝．LG〟という拡張子が付いています．

リスト1

```
100 '
110 '     GRAPH SAMPLE PROGRAM
120 '
130 XN.LG=12
140 XM.LG=4
150 GOSUB *INIT.LG
160 RESTORE *TMP
170 FOR I=0 TO XM.LG-1
180   FOR J=0 TO XN.LG-1
190     READ X.LG(I,J)
200   NEXT
210 NEXT
220 XSTRT.LG=1 : XSTP.LG=1
230 RESTORE *TITLE
240 FOR I=0 TO 5
250   READ A$
260   TITLE.LG%(I)=VAL("&H"+A$)
270 NEXT
280 RESTORE *KOMOKU
290 FOR I=0 TO XM.LG-1
300   READ KOMOKU.LG$(I)
310 NEXT
320 *TITLE
330 DATA 3346,434F,244E,3524,3239,0D0A
340 GOSUB *GRAPH.LG
350 END
360 *TMP
370 DATA .9,1.3,4.2,10.0,14.9,18.4,22.2,23.9,20.0,14.3,8.7,3.7
380 DATA 4.7,5.4,8.4,13.9,18.4,21.5,25.2,26.7,22.9,17.3,12.3,7.4
390 DATA 5.6,5.8,8.3,14.5,19.2,22.8,27,28,24.1,18.3,12.7,7.8
400 DATA 5.7,6.4,9.3,14.2,18.4,22.0,26.7,27.3,23.4,17.8,12.7,8.1
410 *KOMOKU
420 DATA ｾﾝﾀﾞｲ,ﾄｳｷｮｳ,ｵｵｻｶ,ﾌｸｵｶ

40000 *INIT.LG
40010 '
40020 '     SET COLOR & LINE STYLE
40030 '
40040 T1.LG=XN.LG-1 : T2.LG=XM.LG-1
40050 DIM X.LG(T2.LG,T1.LG),CL.LG(13),LS.LG(13),TITLE.LG%(24),KOMOKU.L
G$(T2.LG)
40060 ' LINE COLOR
40070 RESTORE *COLOR.DATA.LG
40080 FOR I.LG=0 TO 13
40090   READ CL.LG(I.LG)
40100 NEXT
40110 ' LINE STYLE
40120 RESTORE *STYLE.DATA.LG
40130 FOR I.LG=0 TO 13
40140   READ LS.LG(I.LG)
40150 NEXT
40160 RETURN
40170 *COLOR.DATA.LG
40180 DATA 1,2,3,4,5,6,7,1,2,3,4,5,6,7
40190 *STYLE.DATA.LG
40200 DATA &HFFFF,&H5555,&H9999,&HE38E,&HF0F0,&HBBBB,&HFFFF,&H5555,&H9
999,&HE38E,&HF0F0,&HBBBB,&HFFFF,&H5555

50000 *GRAPH.LG
50010 '
50020 '     GRAPH
50030 '
50040 '   X.LG(n,m); ﾃﾞｰﾀ
50050 '   XN.LG    ; ﾃﾞｰﾀ ﾉ ｶｽﾞ
50060 '   XM.LG    ; ﾃﾞｰﾀ ﾉ ｼｭﾙｲ
50070 '   XSTRT.LG ; X ｼｮｷﾁ
50080 '   XSTP.LG  ; X ｽﾃｯﾌﾟ
```

```
50090 '   KOMOKU.LG$(m) ; ｺｳﾓｸ
50100 X0.LG=64 : X1.LG=614 : Y0.LG=24 : Y1.LG=182
50110 SCREEN 0,0
50120 CLS 3
50130 GOSUB *FSTEP.LG
50140 GOSUB *KOMOKU.LG
50150 GOSUB *TITLE.LG
50160 VIEW (X0.LG,Y0.LG)-(X1.LG,Y1.LG)
50170 IF XN.LG>20 THEN *PAGE.LG
50180 XC.LG=XN.LG : XN1.LG=0 : XN2.LG=XN.LG-1
50190 GOSUB *DRPG.LG
50200 RETURN
50210 *PAGE.LG
50220 PG.LG=1
50230 XC.LG=20
50240 XN1.LG=0
50250 *DRPGL.LG
50260   IF XN1.LG+20>=XN.LG THEN XN2.LG=XN.LG-1 : GOSUB *DRPG.LG : RET
URN ELSE XN2.LG=XN1.LG+20
50270   GOSUB *DRPG.LG
50280   XN1.LG=XN1.LG+20
50290   PG.LG=PG.LG+1
50300   LOCATE 3,24
50310   PRINT "ｽﾍﾟｰｽ ﾊﾞｰ ﾃﾞ ﾂｷﾞ ﾉ ﾍﾟｰｼﾞ";
50320   *WAIT.LG
50330   IF INKEY$<>" " THEN *WAIT.LG
50340   CLS 3
50350 GOTO *DRPGL.LG
50360 *DRPG.LG
50370 WINDOW (XN1.LG*XSTP.LG,0)-((XN1.LG+XC.LG)*XSTP.LG,FMAX.LG)
50380 GOSUB *SCALE.LG
50390 GOSUB *PRMT.LG
50400 GOSUB *DRAW.LG
50410 LOCATE 68,1
50420 PRINT "PAGE ";PG.LG;
50430 RETURN
50440 *KOMOKU.LG
50450 '
50460 '   PRINT KOMOKU
50470 '
50480 FOR I.LG=0 TO XM.LG-1
50490   LINE (548,I.LG*8+28)-STEP(20,0),CL.LG(I.LG),,LS.LG(I.LG)
50500   LOCATE 72,I.LG+3
50510   PRINT RIGHT$(KOMOKU.LG$(I.LG),8);
50520 NEXT
50530 RETURN
50540 *FSTEP.LG
50550 '
50560 '   PICK UP MAX.DATA AND CALCULATE STEP
50570 '
50580 FMAX.LG=0
50590 FOR J.LG=0 TO XM.LG-1
50600   FOR I.LG=0 TO XN.LG-1
50610     IF FMAX.LG<X.LG(J.LG,I.LG) THEN FMAX.LG=X.LG(J.LG,I.LG)
50620   NEXT
50630 NEXT
50640 DG.LG=INT(LOG(FMAX.LG)/LOG(10))
50650 SCC.LG=INT(FMAX.LG/10^DG.LG)+1
50660 FMAX.LG=SCC.LG*10^DG.LG
50670 RETURN
50680 *SCALE.LG
50690 '
50700 '    DRAW SCALE
50710 '
50720 LINE(0,FMAX.LG)-STEP(XSTP.LG*XN.LG,0)
50730 LINE(XN1.LG*XSTP.LG,FMAX.LG)-STEP(0,-FMAX.LG)
50740 FOR I.LG=XN1.LG TO XN2.LG+1
50750   LINE(XSTP.LG*I.LG,FMAX.LG)-STEP(0,-FMAX.LG/50)
50760 NEXT
50770 IF SCC.LG>=5 THEN FST.LG=1 ELSE FST.LG=.5
50780 FOR I.LG=0 TO FMAX.LG STEP FST.LG*10^DG.LG
50790   LINE(XN1.LG*XSTP.LG,FMAX.LG-I.LG)-STEP(XSTP.LG*XC.LG/100,0)
50800 NEXT
```

1ページ分のグラフ作成（50370〜50430）

最大目盛の決定（50640〜50670）

```
50810 RETURN
50820 *DRAW.LG
50830 '
50840 '  DRAW LINE
50850 '
50860 FOR K.LG=0 TO XM.LG-1
50870   POINT (XN1.LG,FMAX.LG-X.LG(K.LG,XN1.LG))
50880   FOR I.LG=XN1.LG TO XN2.LG
50890     LINE -(XSTP.LG*I.LG,FMAX.LG-X.LG(K.LG,I.LG)),CL.LG(K.LG),,LS
.LG(K.LG)
50900   NEXT
50910 NEXT
50920 RETURN
50930 *PRMT.LG
50940 '
50950 '   PRINT  PARAMETER
50960 '
50970 FOR I.LG=0 TO SCC.LG STEP FST.LG
50980   LOCATE 0,22-INT(20*I.LG/SCC.LG+.5)
50990   PRINT USING '####.##';I.LG*10^DG.LG;
51000 NEXT
51010 IF XC.LG>=10 THEN PRST.LG=2 ELSE PRST.LG=1 ←─目盛のステップ判定
51020 IF (XN2.LG-XN1.LG)<=20 THEN XT.LG=XN2.LG-XN1.LG ELSE XT.LG=XC.LG
51030 FOR I.LG=0 TO XT.LG STEP PRST.LG
51040   LOCATE INT(70*I.LG/XC.LG+5.5),23
51050   PRINT USING '####';XSTRT.LG+(I.LG+XN1.LG)*XSTP.LG
51060 NEXT
51070 RETURN
51080 *TITLE.LG
51090 '
51100 '  PRINT TITLE
51110 '
51120 PT.LG=0
51130 *TITLE.LOOP.LG
51140   IF (TITLE.LG%(PT.LG)=&HD0A) OR (PT.LG=24) THEN RETURN
51150   PUT (32+PT.LG*16,0),KANJI(TITLE.LG%(PT.LG))
51160   PT.LG=PT.LG+1
51170 GOTO *TITLE.LOOP.LG
```

このプログラムのｙ軸の目盛の取り方，数字の振り方などは，ほとんど今まで説明した通りです．全データ中の最大値より大きい$a \times 10^n$を求め，それをｙ軸の最大値としています．目盛はａが６以上のときは1×10^nおき，ａが５以下のときは0.5×10^nおきに振っています．数字は目盛の最大数が11なので，すべてに振っています．また，ワールド座標のｙ方向の値は０から最大目盛まで，最大目盛の値からデータの値を引くことにより，実際のワールド座標を得ています．

ｘ軸の処理はｙ軸の処理とだいぶ異なっています．このプログラムでは，ｘの値は初期値と増分により与えられています．つまり，ｘの値は等間隔な訳です．このような折れ線グラフは，多くの場合，線の角度やつながり方からデータの傾向を読み取ります．そのため，ｘ軸方向にデータを詰め込みすぎると，何が何だか分からなくなってしまいます．そこで，データの数を区切り，ページに分けることにしました．リスト１のプログラムでは，データを20個ごとに区切っています．但し，参考のため21個目のデータも表示しています．また，データの総数が20個未満のときは，その総数で幅をいっぱいに使います．これらはデータの総数 XN に応じて判断され，全データのうちから XN1から XN2の間のデータを表示するサブルーチン＊DRPG によりグラフを表示します．

データの表示が複数ページに渡るときは，スペース・バーを押すことによりページが更新されます．

折れ線グラフの最も重要な部分である折線は，＊DRAW というサブルーチンで描きます．折れ線グラフは，Xの変化に伴うデータの変化を折れ曲った１本の線でつなぐのですから，LINE 文は始点を省略して，前の線の終点を始点とするモードで使います．そのため，折れ線を描くループの前で，POINT 文により最初の線の始点を定めています．

また，PC-8801では簡単に点線を引けるので，LINE 文で色の指定と共にその指定もします．配列 CL が色，配列 LS がライン・スタイルを指定します．ライン・スタイルは16ビットの整数（－32368～32767）の２進のビット・パターンで指定します．従来のパーソナル・コンピュータで描いたグラフは，データの区別を色だけ異なる実線で行っていたのに対し，点線が使用できるというのは大変便利です．特にプリンタでハード・コピーをとるときに，色が同じでもデータの区別ができますから，複数のデータを同時に表示しても十分実用になるようになった訳です．図２に，指定する16進数とライン・スタイルとの関係を示しておきます．

16進数　ライン・スタイル

FFFF
F00F
C3C3
9999
5555

FEFE
FCFC
F8F8
0707
0303
0101
EEEE
1111

F6F6
FFCC
FCCC

図２　指定する数とライン・スタイルの関係

☆サブルーチンの使い方

では，このプログラムを使うための，データの渡し方について説明しましょう．このプログラムに渡す変数は以下の通りです（拡張子〝．LG〃は省略しています）．

○XN

１種類のデータについてのデータの数

○XM

データの種類の数

○X(XM－1，XN－1)

データ

○XSTRT

X の初期値

○XSTP

X の増分

○TITLE% (24)

タイトルの漢字コード

以上の，配列変数の添字は0からです．また，TITLE% (24) の要素は25個ありますが，実際に表示されるのは24文字までです．詳しくは，5．3．2〝漢字の導入〟を読んでください．但し，漢字 ROM を装備していないシステムを使用の場合は，もちろん漢字タイトルの表示はされません．

画面のコード・コピーは COPY キーにより行えますが，プログラム中で行いたい場合は，サブルーチン＊DRPG．LG の RETURN 文の前に挿入してください．

5.2.2 円グラフ

円グラフは他のグラフと違い，データの各要素の全体に対する比率を扇形の角度で表示するグラフです．円グラフを描くプログラムをリスト2に示します．100行台が呼び出しのサンプル・プログラム，40000行台が PAINT 文で使用するタイル・パターンを設定しているデータ設定サブルーチンです．50000行台はグラフを描くサブルーチンです．このサブルーチンで使っている変数名，ラベル名にはすべて〝．CG〟という拡張子が付けてあります．但し，説明においては拡張子は省略します．

リスト2

```
100 '
110 ' CIRCLE GRAPH SAMPLE PROGRAM
120 '
130 XN.CG=7
140 GOSUB *INIT.CG
150 RESTORE *TITLE
160 FOR I=0 TO 11
170   READ A$
180   TITLE.CG%(I)=VAL('&H'+A$)
190 NEXT
200 *TITLE
210 DATA 2336,376E,4A2C,2121,4764,3E65,3662,335B,4662,4C75,0D0A
220 RESTORE *URIAGE
230 FOR I=0 TO XN.CG-1
240   READ X.CG(I)
250 NEXT
260 RESTORE *KOMOKU
270 FOR I=0 TO XN.CG-1
280   READ KOMOKU.CG$(I)
290 NEXT
300 GOSUB *CIRGR.CG
310 IF ERFLG.CG=-1 THEN PRINT 'ERROR'
320 END
330 *KOMOKU
340 DATA VTR,'ﾊﾟｰｿﾅﾙ ｺﾝﾋﾟｭｰﾀﾞ',ｽﾃﾚｵ,ﾃﾚﾋﾞ,ｸｰﾗｰ,ﾗｼﾞｶｾ,ｿﾉﾀ
350 *URIAGE
360 DATA 20502,4632,8910,4587,16687,3958,3359
```

```
40000 *INIT.CG
40010 '
40020 '  SET TILE PATTERN
40030 '
40040 T.CG=XN.CG-1
40050 DIM X.CG(XN.CG),KOMOKU.CG$(T.CG),PTILE.CG$(19),TITLE.CG%(24)
40060 RESTORE *TILE.DATA.CG
40070 FOR I.CG=0 TO XN.CG-1
40080   PTILE.CG$(I.CG)=""
40090   FOR J.CG=0 TO 11
40100     READ PT.CG$
40110     PTILE.CG$(I.CG)=PTILE.CG$(I.CG)+CHR$(VAL("&H"+PT.CG$))
40120   NEXT
40130 NEXT
40140 *TILE.DATA.CG
40150  DATA 00,FF,00,00,FF,00,00,FF,00,00,FF,00
40160  DATA FF,00,00,AA,00,00,FF,00,00,AA,00,00
40170  DATA 00,00,AA,00,00,55,00,00,AA,00,00,55
40180  DATA CC,CC,00,33,33,00,CC,CC,00,33,33,00
40190  DATA FF,00,FF,33,00,33,FF,00,FF,33,00,33
40200  DATA 84,00,84,21,00,21,88,00,88,22,00,22
40210  DATA 81,81,81,42,42,42,24,24,24,18,18,18
40220  DATA 08,08,00,08,08,00,FF,FF,00,08,08,00
40230  DATA 00,00,88,00,00,44,00,00,44,00,00,88
40240  DATA 82,82,00,28,28,00,28,28,00,82,82,00
40250  DATA 00,AA,AA,00,AA,AA,00,55,55,00,55,55
40260  DATA 00,3C,00,00,42,00,00,3C,00,00,00,00
40270  DATA 00,FF,00,00,81,00,00,81,00,00,FF,00
40280  DATA 82,82,00,38,38,00,38,38,00,82,82,00
40290  DATA 00,81,81,00,42,42,00,24,24,00,FF,FF
40300  DATA 00,00,88,00,00,00,00,00,22,00,00,00
40310  DATA FF,00,FF,88,00,88,FF,00,FF,11,00,11
40320  DATA 00,F0,00,00,F0,00,00,0F,00,00,0F,00
40330  DATA 00,08,08,00,10,10,00,3E,3E,00,04,04
40340  DATA 88,00,00,FF,00,00,88,00,00,FF,00,00
40350 RETURN

50000 *CIRGR.CG
50010 '
50020 '    CIRCLE GRAPH
50030 '
50040 '     XN.CG    ; ﾃﾞｰﾀ ﾉ ｶｽﾞ
50050 '     X.CG(n)  ; ﾃﾞｰﾀ
50060 '     KOMOKU.CG$(n)  ; ｺｳﾓｸ
50070 IF XN.CG>20 THEN ERFLG.CG=-1 : RETURN ELSE ERFLG.CG=0
50080 X.CG(XN.CG)=0
50090 SCREEN 0,0
50100 CLS 3
50110 GOSUB *TITLE.CG
50120 '
50130 '   SORT
50140 '
50150 FOR I.CG=0 TO XN.CG-1
50160   FOR J.CG=0 TO I.CG
50170     IF X.CG(I.CG)>X.CG(J.CG) THEN SWAP X.CG(I.CG),X.CG(J.CG) : S
WAP KOMOKU.CG$(I.CG),KOMOKU.CG$(J.CG)
50180   NEXT
50190 NEXT
50200 '
50210 '   DRAW GRAPH
50220 '
50230 PI2.CG=3.14159*2
50240 VIEW (32,35)-(332,185)
50250 WINDOW(-1,-1)-(1,1)
50260 XSUM1.CG=0                          ┐
50270 FOR I.CG=0 TO XN.CG-1               │
50280   XSUM1.CG=XSUM1.CG+X.CG(I.CG)      ├ 合計の計算
50290 NEXT                                ┘
50300 CIRCLE(0,0),1,7
50310 XSUM2.CG=X.CG(0)
50320 LINE(0,0)-(0,-1),7
50330 FOR I.CG=1 TO XN.CG
```

```
50340     AG.CG=PI2.CG*XSUM2.CG/XSUM1.CG
50350     CX.CG=SIN(AG.CG)   }ラインの終点座標の決定
50360     CY.CG=-COS(AG.CG)  }
50370     LINE(0,0)-(CX.CG,CY.CG),7
50380     AG.CG=PI2.CG*(XSUM2.CG-.5*X.CG(I.CG-1))/XSUM1.CG
50390     CX.CG=.9*SIN(AG.CG)   }ペイント座標の決定
50400     CY.CG=-.9*COS(AG.CG)  }
50410     PAINT (CX.CG,CY.CG),PTILE.CG$(I.CG-1),7
50420     XSUM2.CG=XSUM2.CG+X.CG(I.CG)←― 部分和の計算
50430 NEXT
50440 SCREEN 0,0
50450 FOR I.CG=0 TO XN.CG-1                      }
50460     LINE (400,I.CG*8+40)-STEP(24,6),7,B      }
50470     PAINT (412,I.CG*8+44),PTILE.CG$(I.CG),7  } 項目の表示
50480     LOCATE 54,I.CG+5                         }
50490     PRINT KOMOKU.CG$(I.CG);                  }
50500 NEXT                                       }
50510 RETURN
50520 *TITLE.CG
50530 '
50540 '    PRINT TITLE
50550 '
50560 PT.CG=0
50570 *TITLE.LOOP.CG
50580     IF (TITLE.CG%(PT.CG)=&HD0A) OR (PT.CG=24) THEN RETURN
50590     PUT (32+PT.CG*16,0),KANJI(TITLE.CG%(PT.CG))
50600     PT.CG=PT.CG+1
50610 GOTO *TITLE.LOOP.CG
```

この章で紹介している各グラフのデータは，適当に作成したもので，特に意味はありません．

円グラフを描くには，まずその前に，データをある程度処理しなければなりません．最初にデータの総数のチェックを行います．これが多すぎると，グラフが非常に細かく分割されて，何のために見易いグラフにしたのか分からなくなってしまいます．このプログラムでは，最大20個までのデータを扱うことにしました．次にデータのソーティング（並べかえ）を行います．円グラフは，普通データを大きい順に表示します．そして，各データの比率を求めるために全データの合計 XSUM 1を求めます．

以上で，データの前処理は終りです．

では，このデータの大きさにより円グラフを描く部分について説明しましょう．N_{88}-BASIC では，CIRCLE 文で円弧の始点，終点を設定できます．このとき，始点，終点の値をマイナスで与えると，円弧だけでなく中心から終点まで直線を描きます．この機能を使うと割と簡単に円グラフを描けそうに思うのですが，実際にやってみると以外に面倒なことが分かります．まず，円グラフは時計の12時の位置が始点であるのに対し，CIRCLE 文は 3 時の位置が始点なのです．さらに，回転方向も逆になっています．そこで，見た目に正方形になるようにビューポートを設定し，その領域を（－1，＋1）－（1，1）になるようにウィンドウを設定し三角関数を使うことにし

ました．

まず，CIRCLE 文で半径1の円を描きます．そして，中心から円周上までの線を引きます．n本めの線を引く角度 AG は，次の様に求まります．

$$AG=2\pi\cdot\sum_{i=0}^{n}X(n)/XSUM1$$

部分和を毎回計算するのは時間の無駄ですから，ループを回るごとに新しいデータを1つずつ XSUM 2 という変数に加え，部分和としています．XSUM2には，初めX(0) を代入しておき，ループをデータの数だけ回ると，線をすべて描くことができます．このループを回る回数とパラメータの都合により，データを代入しておく配列Xは，要素数を XN＋1 個にしておき，X(XN) は，0にしておかなければなりません．

始点，終点の座標は，三角関数を使い角度から求めます．このとき，y 座標は上の方が－1になっているため，符号を反対にしておきます．

これで線は引けました．後は，ペイントする始点の座標を計算しなければなりません．ここでは，角度が2本の線の角度の中間値であり，中心からの距離が0.9になる位置にしました．中間の角度は，XSUM 2 から現在着目しているデータを1／2したものを引き，後は前と同じように計算して得ています．そして，この点から PTILE$ という配列のパターンでペイントしています．このパターンは4章のタイル・パターン・ジュネレータで作成したものです．

円グラフ本体はこれで描き終わった訳ですが，まだ，各データの項目名を書かなければなりません．これは，円グラフの左に，中を円グラフと同じパターンでペイントした，小さな四角形を描き，その横に KOMOKU$ という配列の内容を出力するだけです．この小さな四角形は，640×200モードのスクリーン座標で位置を指定していますから，もし640×400モードを使用する場合は，書き直さなければなりません．

グラフのタイトルについては，5．3．2〝漢字の導入〟を見てください．

☆サブルーチンの使い方

このサブルーチンを使うためには，以下の配列，変数に必要なデータを代入し，引き渡さなければなりません．(拡張子〝．CG〟は省略してあります．)．

○**XN**

データの数

○**X(XN)**

データ．但し，X(XN) は0でダミー

○**KOMKU$(XN－1)**

各データの項目名

○**TITLE%(24)**

タイトルの漢字コード

このサブルーチンを呼んだとき，データの数が21個以上だとERFLGを－1にセットし，何もしないでリターンします．この数字は，項目を表示する行数の都合で決定したものですが，実用上からもデータ個数の多すぎる円グラフはあまり感心しません．多くなりそうなときは，細かいデータをひとまとめにし，〝その他〟という1つの項目にすべきでしょう．

画面のハード･コピーは，コピー･キーで行えますが，プログラム中で行いたい場合は，＊TITLEの直前のRETURN文の前に入れてください．このプログラムで使用しているタイル・パターンは，ハード・コピーの使用を考えて，色はもちろんのことパターンもすべて違うものを用いています．

5.2.3 レーダー・チャート

レーダー・チャートとは，折れ線グラフの右端と左端をつなぎ，x軸を点とした様なグラフです．折れ線グラフと違う点は，y軸に相当する放射状の直線に対応するx軸の値は，必ずしも連続した数値ではないということです．

プログラムをリスト3に示します．100行台が呼び出しのサンプル・プログラム，40000行台がライン･スタイルと色を設定するデータ設定サブルーチン，50000行台がグラフを描くルーチンです．これらのサブルーチンに使われている変数名，ラベル名には，拡張子〝．RC〟が付けられています．

リスト3

```
100 '
110 '   RADAR CHART SAMPLE
120 '
130 XN.RC=5
140 XM.RC=3
150 GOSUB *INIT.RC
155 RESTORE *SHOYURITSU
160 FOR I=0 TO XM.RC-1
170   FOR J=0 TO XN.RC-1
180     READ X.RC(I,J)
190   NEXT
200 NEXT
210 RESTORE *TITLE
220 FOR I=0 TO 6
230   READ A$
240   TITLE.RC%(I)=VAL('&H'+A$)
250 NEXT
260 *TITLE
270 DATA 472F,4265,4A4C,3D6A,4D2D,4E28
280 DATA 0D0A
290 RESTORE *KOMOKU
300 FOR I=0 TO XN.RC-1
310   READ KOMOKU1.RC$(I)
320 NEXT
330 FOR I=0 TO XM.RC-1
340   READ KOMOKU2.RC$(I)
350 NEXT
360 GOSUB *RADAR.CHART.RC
370 IF ERFLG.RC=-1 THEN PRINT 'ERROR'
380 END
390 *KOMOKU
400 DATA VTR,ｽﾃﾚｵ,ｸｰﾗｰ,ｼﾞﾄﾞｳｼｬ,'ﾊﾟｰｿﾅﾙ ｺﾝﾋﾟｭｰﾀｰ'
410 DATA '10 ﾀﾞｲ','20 ﾀﾞｲ','30 ﾀﾞｲ'
420 *SHOYURITSU
430 DATA 5,30,25,2,12
440 DATA 12,40,30,15,20
450 DATA 15,35,40,30,10
```

```
40000 *INIT.RC
40010 '
40020 '  SET LINE STYLE & COLOR
40030 '
40040 T1.RC=XM.RC-1 : T2.RC=XN.RC-1
40050 DIM X.RC(T1.RC,T2.RC),KOMOKU1.RC$(T2.RC),KOMOKU2.RC$(T2.RC)
40060 DIM TITLE.RC%(24),LS.RC(9),CL.RC(9)
40070 RESTORE *STYLE.RC
40080 FOR I.RC=O TO 9
40090   READ LS.RC(I.RC)
40100 NEXT
40110 RESTORE *COLOR.RC
40120 FOR I.RC=O TO 9
40130   READ CL.RC(I.RC)
40140 NEXT
40150 RETURN
40160 *STYLE.RC
40170 DATA &HFFFF,&H5555,&H9999,&HE38E,&HF0F0,&HBBBB,&HFFFF,&H5555,&H9
999,&HE38E
40180 *COLOR.RC
40190 DATA 1,2,3,4,5,6,7,1,2,3

50000 *RADAR.CHART.RC
50010 '
50020 '    RADAR CHART
50030 '
50040 '   X.RC(n,m) ; ﾃﾞｰﾀ
50050 '   XM.RC     ; ｼﾞｹﾞﾝ
50060 '   XN.RC     ; ﾃﾞｰﾀ ﾉ ｶｽﾞ
50070 '   KOMOKU1$(n); ｼﾞｸ ﾉ ｺｳﾓｸ
50080 '   KOMOKU2$(n); ｾﾝ ﾉ ｺｳﾓｸ
50090 IF (XM.RC>10) OR (XN.RC>10) OR (XM.RC<3) THEN ERFLG.RC=-1 : RETU
RN ELSE ERFLG.RC=0
50100 SCREEN 0,0
50110 CLS 3
50120 GOSUB *TITLE.RC
50130 PI2.RC=3.14159*2
50140 VIEW (48,32)-(351,183)
50150 GOSUB *MAX.RC
50160 MAX.RC=5*UNIT.RC
50170 WINDOW (-MAX.RC,-MAX.RC)-(MAX.RC,MAX.RC)
50180 GOSUB *SCALE.RC
50190 GOSUB *DRAW.RC
50200 GOSUB *KOMOKU.RC
50210 RETURN
50220 *MAX.RC
50230 '
50240 '   PICK UP MAX.DATA
50250 '
50260 RMAX.RC=0
50270 FOR I.RC=0 TO XM.RC-1
50280   FOR J.RC=0 TO XN.RC-1
50290     IF X.RC(I.RC,J.RC)>RMAX.RC THEN RMAX.RC=X.RC(I.RC,J.RC)
50300   NEXT
50310 NEXT
50320 UNIT.RC=RMAX.RC/5                              ┐
50330 DG.RC=INT(LOG(UNIT.RC)/LOG(10))                ├ 目盛幅の決定
50340 UNIT.RC=INT(UNIT.RC/10^DG.RC+1)*10^DG.RC       ┘
50350 RETURN
50360 *SCALE.RC
50370 '
50380 '    DRAW SCALE
50390 '
50400 FOR I.RC=0 TO XN.RC-1                          ┐
50410   ANG.RC=PI2.RC*I.RC/XN.RC                     │
50420   GOSUB *CRD.RC                                ├ 放射軸の表示
50430   LINE (0,0)-(X.RC*MAX.RC,Y.RC*MAX.RC),7       │
50440 NEXT                                           ┘
```

```
50450 FOR I.RC=1 TO 5
50460   ANG.RC=0
50470   GOSUB *CRD.RC
50480   POINT (X.RC*I.RC*UNIT.RC,Y.RC*I.RC*UNIT.RC)
50490   FOR J.RC=1 TO XN.RC
50500     ANG.RC=PI2.RC*J.RC/XN.RC
50510     GOSUB *CRD.RC
50520     LINE -(X.RC*I.RC*UNIT.RC,Y.RC*I.RC*UNIT.RC),7,,&H9999
50530   NEXT
50540 NEXT
50550 RETURN
50560 *CRD.RC
50570 '
50580 '    CALCULATE CRD.
50590 '
50600 X.RC=SIN(ANG.RC)
50610 Y.RC=-COS(ANG.RC)
50620 RETURN
50630 *DRAW.RC
50640 '
50650 '  DRAW LINE
50660 '
50670 FOR I.RC=0 TO XM.RC-1
50680   ANG.RC=0
50690   GOSUB *CRD.RC
50700   POINT (X.RC*X(I.RC,0),Y.RC*X.RC(I.RC,0))
50710   FOR J.RC=1 TO XN.RC
50720     ANG.RC=PI2.RC*J.RC/XN.RC
50730     GOSUB *CRD.RC
50740     LINE -(X.RC*X.RC(I.RC,J.RC MOD XN.RC),Y.RC*X.RC(I.RC,J.RC MO
D XN.RC)),CL.RC(I.RC),,LS.RC(I.RC)
50750   NEXT
50760 NEXT
50770 RETURN
50780 *KOMOKU.RC
50790 '
50800 '    PRINT KOMOKU
50810 '
50820 FOR I.RC=0 TO XN.RC-1
50830   ANG.RC=PI2.RC*I.RC/XN.RC
50840   GOSUB *CRD.RC
50850   CX.RC=INT(24.5+X.RC*21)
50860   CY.RC=INT(13.5+Y.RC*11)
50870   LOCATE CX.RC,CY.RC
50880   PRINT I.RC+1;
50890   LOCATE 50,I.RC+2
50900   PRINT I.RC+1;" ";KOMOKU1.RC$(I.RC);
50910 NEXT
50920 SCREEN 0,0
50930 FOR I.RC=0 TO XM.RC-1
50940   LINE(400,100+I.RC*8)-STEP(16,0),CL.RC(I.RC),,LS.RC(I.RC)
50950   LOCATE 54,12+I.RC
50960   PRINT KOMOKU2.RC$(I.RC);
50970 NEXT
50980 LOCATE 55,24
50990 PRINT UNIT.RC;"/div";
51000 RETURN
51010 *TITLE.RC
51020 '
51030 '    PRINT TITLE
51040 '
51050 PT.RC=0
51060 *TITLE.LOOP.RC
51070   IF (TITLE.RC%(PT.RC)=&HD0A) OR (PT.RC=24) THEN RETURN
51080   PUT (32+PT.RC*16,0),KANJI(TITLE.RC%(PT.RC))
51090   PT.RC=PT.RC+1
51100 GOTO *TITLE.LOOP.RC
```

目盛 (50450–50540)

項目のキャラクタ座標の算出 (50850–50860)

このグラフも円グラフと同じ様に，正方形のビューポートを，

(−〈最大目盛〉，−〈最大目盛〉)−(〈最大目盛〉，〈最大目盛〉)

に設定しています．このことにより，極座標を直交座標に変換する計算が簡単になります．

レーダー・チャートはm個の項目があった場合，正m角形のクモの巣のようになります．このクモの巣の同心の正m角形は，単なる目盛線で，このプログラムでは常に5目盛になっています．このプログラムでは，最大の要素の値を5で割り，その結果を，切りのいい値に切り上げているため，目盛の幅は中途半端な値にはなりません．切りのいい値にするための計算は，折れ線グラフのそれと同じです．また，数値は各目盛に振るとグラフがごちゃごちゃになってしまうため，1目盛あたりの数値がいくつになるかを，グラフの横に出力しています．

グラフの軸は，中心から正m角形の各頂点までの直線を引くだけで，角度は2πをm等分するだけですから，円グラフより簡単です．目盛線は，折れ線グラフの折れ線と同様に，POINT 文で最初の線の始定を設定し，LINE 文の始点を省略して正m角形を描いていきます．この辺の手法は，2章の多角形の描き方で解説した方法とほとんど同じです．目盛線はグラフを見易くするためライン・スタイルを用いて点線にしています．

データを表す折れ線も，目盛線と同じ様に，最初に POINT 文で始点を設定し，LINE 文の始点を省略したモードで引いていきます．POINT 文や LINE 文で用いる座標は次のようにして求めます．

```
X=X(n)*SIN (ANG)
Y=−X(n)*COS (ANG)
```

nは0から始まるデータの添字，ANG はそのときの角度で，12時の角度から時計回りに進みます．この計算のx座標の SIN の計算，y座標の−COS の計算は，目盛線を引くのにも使用するので，サブルーチンにしています．

データを表す折れ線の最後の終点は，その折れ線の最初の線の始点につながります．そのため，最後の線の終点が最初のデータの値となるように，データの配列を参照するときの添字は，データの総数 XN で MOD を取ったものにしています．

レーダー・チャートは前にも説明した通り，他のグラフのY軸に相当するものが複数あり，それが必ずしも連続した数値である必要がないため，これも項目とすることができます．そのため，線の区別と合わせて，2種類の項目を持つことができます．例えば，ライン・スタイルや色で区分されるデータを20代，30代，40代と分け，放射状の軸で区分されるデータを VTR，自動車，ステレオ，クーラーなどの所有率とすれば，それらの所有率と年代の関係などを視覚的に表すことが可能です．

このサブルーチンでは，20代，30代といった線による項目を，KOMOKU2$ という配列，VTR，自動車といった，軸による項目を KOMOK1$ という配列に持っています．これらの画面への出

力は，＊KOMOKU というサブルーチンにより行っています．軸の項目は，レーダー・チャート上に表示すると，グラフが見にくくなってしまうため，グラフの各頂点に数字を振り，その横に数字に対応する項目を表示するという方法を取りました．この数字を振るのは，データや目盛線と同じ様に，三角関数を使って計算しています．このときキャラクタ座標系の縦・横比が1でないので注意が必要です．LOCATE 文用に座標を計算している式中の，21，11という数字は，それぞれキャラクタ座標でのレーダー・チャートのX方向，Y方向の半径です．24.5，13.5は，同じく中心座標です．小数点以下に0.5が付いているのは，四捨五入するためです．

線による項目は，線のサンプルと共にグラフの横に表示します．このとき線のサンプルを描く前にSCREEN　0，0を実行し，画面のビューポートやウィンドウを初期状態に戻しておく必要があります．

この表示は640×200モードのため，640×400モードを使う場合は，この LINE 文の座標も書き直す必要があります．また，このサブルーチンで，目盛線が1目盛あたりの数値の表示も行っています．

☆**サブルーチンの使い方**

このサブルーチンを使うためには，以下の配列，変数に必要なデータを代入し，引き渡さなければなりません（拡張子〝．RC〞は省略してあります）．

○XN

　1種類のデータについてのデータの数（n角形のnに相当）

○XM

　データの種類の数（データを表す線の本数）

○X(XM－1，XN－1)

　データ

○TITLE%（24）

　タイトルの漢字コード

○KOMOKU1$(XN－1)

　各軸についての項目

○KOMOKU2$(XM－1)

　各線についての項目

このサブルーチンは，グラフを見易くするためXM≦10かつ，3≦XN≦10のときのみグラフを描き，それ以外のときは，ERFLG を－1にセットし，何もせず RETURN します．

画面のハード・コピーは，コピー・キーにより行えますが，プログラム中で行いたい場合は，サブルーチンの50210行の RETURN 文の前（＊MAX の直前）に入れてください．

5.2.4 シンボル・グラフ

シンボル・グラフというのは，数量をグラフ上の位置や長さで表すのでなく，１つが一定の数量を持ったシンボルの数で表すものです．このグラフは細かい数値を表すのには用いず，大ざっぱな値を見易い形で表すために使用します．プログラムをリスト４に示します．100行台は呼び出しのサンプル･プログラム，40000行台はデータの設定や配列を宣言するデータ設定サブルーチン，50000行台はグラフを描くサブルーチンです．

リスト4

```
100 '
110 '   SYMBOL GRAPH SAMPLE
120 '
130 XN.SG=15
140 GOSUB *INIT.SG
145 RESTORE *DAISU
150 FOR I=0 TO XN.SG-1
160   READ X.SG(I)
170 NEXT
180 XSTRT.SG=1970 : XSTP.SG=1
190 RESTORE *TITLE
200 FOR I=0 TO 12
210   READ A$
220   TITLE.SG%(I)=VAL("&H"+A$)
230 NEXT
240 GOSUB *SYMGR.SG
250 END
260 *TITLE
270 DATA 3B3A,3648,4D51,256D,255C,2543,2548
280 DATA 2121
290 DATA 3254,4630,4266,3F74,0D0A
300 *DAISU
310 DATA 25,139,320,300,420,330,460,800,1354,2035,2800,2600,3200,4500,
3900

40000 *INIT.SG
40010 '
40020 '  SET PATTERN DATA
40030 '
40040 T.SG=XN.SG-1
40050 DIM X.SG(T.SG),SYM.SG%(147),TITLE.SG%(24)
40060 RESTORE *PAT.DATA.SG
40070 FOR I.SG=0 TO 147
40080   READ SYM.SG%(I.SG)
40090 NEXT
40100 RETURN
40110 *PAT.DATA.SG
40120 DATA 48 , 16 , 0 , 0 , 0 , 0 , 0 , 0
40130 DATA 0 , 0 , 0 , 0 , 0 ,-4065 ,-241 , 248
40140 DATA-4065 , 0 , 0 ,-4065 , 0 , 0 , 2080 ,-225
40150 DATA 252 , 2080 , 0 , 0 , 2080 , 0 ,-255 , 224
40160 DATA-193 ,-1 , 224 , 0 , 0 , 32 , 0 , 0
40170 DATA 2080 ,-225 , 252 , 2080 , 0 , 0 , 2080 , 0
40180 DATA 0 ,-4065 ,-241 , 248 ,-4065 , 0 , 0 ,-4065
40190 DATA 0 , 0 , 0 , 0 , 0 , 0 , 3840 , 128
40200 DATA 0 , 0 , 0 , 0 , 0 , 0 , 0 , 3840
40210 DATA 128 , 0 , 0 , 0 , 0 , 0 , 0 , 0
40220 DATA 3840 , 128 , 0 ,-253 , 240 , 0 ,-253 , 240
40230 DATA 0 ,-253 , 240 , 0 ,-249 , 248 , 0 , 4
40240 DATA 8 , 0 , 4 , 8 , 0 ,-241 , 252 , 0
40250 DATA 8 , 4 , 0 , 8 , 4 , 0 , 0 , 0
40260 DATA 0 ,-225 ,-1 , 254 ,-225 ,-1 , 254 , 0
40270 DATA 0 , 0 ,-225 ,-1 , 254 ,-225 ,-1 , 254
40280 DATA 0 , 0 , 0 ,-225 ,-1 , 254 ,-225 ,-1
40290 DATA 254 , 0 , 0 , 0 , 0 , 0 , 0 , 0
40300 DATA 0 , 0 , 0 , 0 , 0
```

（40120～40300：シンボルのデータ）

ブルドーザー 販売台数　PAGE 1　100

年	台数
1970	300
1971	139
1972	320
1973	200
1974	420
1975	490
1976	460
1977	800
1978	530
1979	235

ｽﾍﾟｰｽｷｰｦ ｵｽﾄ ﾂｷﾞﾉ ﾍﾟｰｼﾞ

```
50000 *SYMGR.SG
50010 '
50020 '    SYMBOL GRAPH
50030 '
50031 '    X(n)    ; ﾃﾞｰﾀ
50032 '    XN      ; ﾃﾞｰﾀ ﾉ ｶｽﾞ
50033 '    XSTRT   ; X ﾉ ｼｮｷﾁ
50034 '    XSTPT   ; X ﾉ ｿﾞｳﾌﾞﾝ
50040 SCREEN 0,0
50050 CLS 3
50060 X0.SG=47 : Y0.SG=31
50070 GOSUB *MAX.SG
50080 GOSUB *TITLE.SG
50090 XN1.SG=0
50100 *LOOP.SG
50110   IF (XN1.SG+10)>=XN.SG THEN XN2.SG=XN.SG-1  : GOSUB *PAGE.SG :
RETURN                      ELSE XN2.SG=XN1.SG+9 : GOSUB *PAGE.SG
50120   XN1.SG=XN1.SG+10
50130   LOCATE 3,24
50140   PRINT "ｽﾍﾟｰｽ ﾊﾞｰ ﾃﾞ ﾂｷﾞ ﾉ ﾍﾟｰｼﾞ";
50150   *WAIT.SG : IF INPUT$(1)<>" " THEN *WAIT.SG
50160   CLS
50170   LINE(X0.SG,Y0.SG)-(639,199),0,BF
50180 GOTO *LOOP.SG
50190 *PAGE.SG
50200   LOCATE 70,0
50210   PRINT "PAGE ";INT(XN1.SG/10)+1
50220 GOSUB *SCALE.SG
50230 GOSUB *PUT.DATA.SG
50240 GOSUB *X.PRMT.SG
50250 RETURN
50260 *MAX.SG
50270 '
50280 ' PICK UP MAX.DATA
50290 '
50300 MAX.SG=0
50310 FOR I.SG=0 TO XN.SG-1
50320   IF X.SG(I.SG)>MAX.SG THEN MAX.SG=X.SG(I.SG)
50330 NEXT
50340 UNIT.SG=MAX.SG/10
50350 DG.SG=INT(LOG(UNIT.SG)/LOG(10))
50360 UNIT.SG=INT(UNIT.SG/(10^DG.SG))                          単位数量の決定
50370 IF UNIT.SG>=5 THEN UNIT.SG=10^(DG.SG+1)
50380 IF (UNIT.SG>=2) AND (UNIT.SG<5) THEN UNIT.SG=5*10^DG.SG
50390 IF (UNIT.SG>=1) AND (UNIT.SG<2) THEN UNIT.SG=2*10^DG.SG
50400 RETURN
50410 *SCALE.SG
50420 '
50430 '    DRAW SCALE
50440 '
50450 XW.SG=48*INT(MAX.SG/UNIT.SG+1.5)
50460 LINE(X0.SG,Y0.SG)-STEP(XW.SG,0)
50470 LINE(X0.SG,Y0.SG)-STEP(0,160)
50480 PUT@(500,8),SYM.SG%,PSET
50490 LOCATE 70,2
50500 PRINT UNIT.SG;
50510 RETURN
50520 *PUT.DATA.SG
50530 '
50540 '    PUT SYMBOL
50550 '
50560 FOR I.SG=XN1.SG TO XN2.SG
50570   Y.SG=16*(I.SG MOD 10)+Y0.SG ←シンボルの数を決定
50580   FOR J.SG=0 TO INT(X.SG(I.SG)/UNIT.SG-.5)
50590     PUT@(X0.SG+48*J.SG+1,Y.SG+1),SYM.SG%,PSET
50600   NEXT
50610   LOCATE (J.SG+1)*6+1,5+2*(I.SG MOD 10)
50620   PRINT X.SG(I.SG);
50630 NEXT
50640 RETURN
```

```
50650 *X.PRMT.SG
50660 '
50670 '  PRINT X PARAMETER
50680 '
50690 FOR I.SG=XN1.SG TO XN2.SG
50700   LOCATE 0,4+2*(I.SG MOD 10)
50710   PRINT USING "#####";XSTRT.SG+XSTP.SG*I.SG;
50720 NEXT
50730 RETURN
50740 *TITLE.SG
50750 '
50760 '  PRINT TITLE
50770 '
50780 PT.SG=0
50790 *TITLE.LOOP.SG
50800   IF (TITLE.SG%(PT.SG)=&HD0A) OR (PT.SG=24) THEN RETURN
50810   PUT (32+PT.SG*16,0),KANJI(TITLE.SG%(PT.SG))
50820   PT.SG=PT.SG+1
50830 GOTO *TITLE.LOOP.SG
```

グラフの構成は，折れ線グラフに似ていて，データの数によりページを更新する方法を用いています．違うところは，線を引くのでなくシンボルを並べること，縦横が入れ換わっているということです．

シンボルを画面に表示するのは，PUT@によって行います．PUTするデータは，データ設定サブルーチンでDATA文により配列に読み込んでいます．このプログラムは，PUT@を使っているため，VIEWやWINDOW命令は使っていません．そのため，640×400モードを使う場合，かなり変更するところがあります．

では，プログラムについて，説明しましょう．まず，例によってデータの最大値を求めます．この最大値が，グラフからはみ出さないように，シンボル1個あたりの単位数量を決定します．シンボルの大きさは，48×16ドットとし，横に最大10個までとしています．この範囲に収まるように切りのいい単位数量を決定しますが，ここで注意することが単位数量に用いる切りのいい値が，今まで出てきた切りのいい数値より，さらに切りがいい必要があるからです．例えば，シンボル1個が70とか90といった数量を表していたらあまりいいグラフとはいえません．そこで，このプログラムでは，先頭の数字が，1，2，5となるようにIF文で単位数量を定めています．

この様にして決定された最大値と単位数量から，横軸を引きます．横軸は最大値のときPUTされるシンボル列の長さだけ，縦軸は常時データ10個分引いています．このグラフでは，数量をシンボル数で表すため，横軸には目盛や数字は振っていません．縦軸はシンボルが縦16ドットですから，2行置きに数値だけ振っています．この数値は間隔が一定で，初期値と増分だけ与える型式のため，簡単に振ることができます．

次にシンボルのPUTについて説明しましょう．これには，まずデータの値に応じて，シンボルをいくつ並べるかを計算します．このプログラムでは単位数量でデータを割ったとき，端数は四捨五入し，中途半端なシンボルは表示しません．これは，このグラフの性質が大まかなデータの表示であるからです．それに，シンボルの横に念のため数値を表示しているので，問題はないと思います．もし，どうしてもシンボルを途中まで表示させたい場合は，LINE文のBFオプションで黒い四角形を描けば良いでしょう．

さて，並べる数が分かれば，後はループを回してその数だけ PUT すれば良い訳です．それを10回，あるいはデータの数だけ繰り返すだけです．

このデータの数に応じて変化するループの回数は，このサブルーチンの最初で決定されます．XN1 とXN2が，実際に表示するデータの最初と最後の添字です．また，ページが複数に渡るときは，スペース・バーで次のページへ進みます．

☆サブルーチンの使い方

このサブルーチンを使うには，以下の配列，変数にデータを代入し，引き渡す必要があります（拡張子〝.SG″は省略してあります）．

○XN
　データの数
○X(XN－1)
　データ
○XSTRT
　X軸（縦軸）の初期値
○XSTP
　X軸のステップ
○TLTLE%（24)
　タイトルの漢字コード

画面のハード・コピーは，コピー・キーで行えますが，プログラム中で行いたい場合は，サブルーチン＊PAGE の RETURN 文の前に入れてください．

☆シンボルのデータ

このサブルーチンを使う前に，PUT するシンボルのデータを配列 SYM%(147) に代入しておく必要があります．リスト 4 のサンプル・プログラムでは，データ設定サブルーチンで，DATA 文から読んでいます．

この DATA 文は，4 章のシンボル・ジェネレータで作成したものです．このプログラムは，DATA 文の作成とデータ・ファイルの作成を行えますが，データ・ファイルを作成したときのデータ入力サブルーチンは，リスト 5 の様になります．まず，FILE.NAM$ という文字変数にファイル名を代入しておきます．シンボルを別のデータに変更するときは，FILE.MAM$ にファイル名を代入した後，＊INIT2を呼んでください．これは，配列宣言の 2 重定義を防ぐためです．

リスト5

```
40000 *INIT.SG
40010 '
40020 '  SET PATTERN DATA
40030 '
40040 T.SG=XN.SG-1
40050 DIM X.SG(T.SG),SYM.SG%(147),TITLE.SG%(24)
40060 *INIT2.SG
40070 OPEN FILE.NAM.SG$ FOR INPUT AS #1
40080 FOR I.SG=0 TO 147
40090   INPUT #1,SYM.SG%(I.SG)
40100 NEXT
40110 CLOSE #1
40120 RETURN
```

ここでは，GET したデータをシーケンシャル・ファイルとして扱いましたが，速度の点で気になる方は，4章の〝Gエディタ〟で使っている BLOAD／BSAVE の手法を用いても構いません．

5.2.5 対数グラフ

対数グラフというのは，軸に対数目盛を振ってあるグラフです．対数グラフには，x軸，y軸共に対数目盛の両対数グラフ，一方の軸だけ対数目盛の片対数グラフがありますが，ここでは両対数グラフを取り上げました（3章で紹介した対数関数を思い出してください）．

対数グラフは次のような時によく用いられます．

○値の最大値と最小値の間に数桁のひらきがある．

○値の対数をとったときに，それぞれの値の関係が見えてくるようなデータ．

両対数グラフを描くサブルーチンをリスト6に示します．100行台は呼び出しのサンプル・プログラム，40000行台は配列の宣言と常用対数の定義を行っている設定ルーチン，50000行台はグラフを描くサブルーチンです．もし，片対数グラフを描きたい場合は，このプログラムと折れ線グラフを参考にすればよいでしょう．

リスト6

```
100 '
110 '  LOG-LOG GRAPH SAMPLE
120 '
130 XN.LGG=20
140 GOSUB *INIT.LGG
150 RESTORE *DAT
160 FOR I=0 TO XN.LGG-1
170   READ X.LGG(I,1)
180   READ X.LGG(I,0)
190 NEXT
200 RESTORE *TITLE
210 FOR I=0 TO 7
220   READ A$
230   TITLE.LGG%(I)=VAL("&H"+A$)
240 NEXT
250 GOSUB *LOGGR.LGG
260 END
270 *TITLE
280 DATA 467E,3D50,4E4F,4A51,3439,4643,402D,0D0A
290 *DAT
300 DATA 2.24315 , 2.48313 , 2.73257 , 3.02817 , 2.79508 , 3.10591 , 5
.46244 ,  9.15982 , 1.2119 , 1.63458 , 13.4645 , 234.951 , 5.14676 , 8
.05935 , 3.31635 ,  3.83689 , 25.6592 , 32988.5 , 19.5211 , 2738.44
310 DATA 10.9913 , 86.1938 , 1.02278 , 1.51392 , 24.4122 , 19896.2 , 1
.00579 ,  1.50353 , 23.3725 , 13052.7 , 1.14381 , 1.59006 , 19.2032 ,
2407.19 , 8.81055 , 35.6009 , 6.21836 , 12.4451 , 14.8532 , 412.597
```

```
40000 *INIT.LGG
40010 '
40020 '    DEFINE ARRAY
40030 '
40040 DEF FNLOG10(X)=LOG(X)/2.30259 ←―常用対数を求める関数
40050 T.LGG=XN.LGG-1
40060 DIM X.LGG(T.LGG,1),TITLE.LGG%(24)
40070 RETURN

50000 *LOGGR.LGG
50010 '
50020 '    LOG-LOG GRAPH
50030 '
50040 ' X(n,0)   ; X
50050 ' X(n,1)   ; Y
50060 ' XN       ; ﾃﾞｰﾀ ﾉ ｶｽﾞ
50070 SCREEN 0,0
50080 CLS 3
50090 GOSUB *TITLE.LGG
50100 X0.LGG=48 : Y0.LGG=183
50110 X1.LGG=607 : Y1.LGG=16
50120 VIEW (X0.LGG,Y1.LGG)-(X1.LGG,Y0.LGG)
50130 GOSUB *SCALE.LGG
50140 WINDOW (XMIN.LGG,YMIN.LGG)-(XMAX.LGG,YMAX.LGG)
50150 GOSUB *XLOGSC.LGG
50160 GOSUB *YLOGSC.LGG
50170 GOSUB *PRMT.LGG
50180 GOSUB *PLOT.LGG
50190 RETURN
50200 *SCALE.LGG
50210 '
50220 '    DRAW SCALE
50230 '
50240 K.LGG=0
50250 GOSUB *PICKUP.MAX.LGG
50260 GOSUB *PICKUP.MIN.LGG
50270 XMAX.LGG=INT(FNLOG10(MAX.LGG))+1
50280 XMIN.LGG=INT(FNLOG10(MIN.LGG))
50290 K.LGG=1
50300 GOSUB *PICKUP.MAX.LGG
50310 GOSUB *PICKUP.MIN.LGG
50320 YMAX.LGG=INT(FNLOG10(MAX.LGG))+1
50330 YMIN.LGG=INT(FNLOG10(MIN.LGG))
50340 RETURN
50350 *PICKUP.MAX.LGG
50360 '
50370 '    PICK UP MAX. DATA
50380 '
50390 MAX.LGG=-1.70141E+38
50400 FOR I.LGG=0 TO XN.LGG-1
50410   IF MAX.LGG<X.LGG(I.LGG,K.LGG) THEN MAX.LGG=X.LGG(I.LGG,K.LGG)
50420 NEXT
50430 RETURN
50440 *PICKUP.MIN.LGG
50450 '
50460 '    PICK UP MIN. DATA
50470 '
50480 MIN.LGG=1.70141E+38
50490 FOR I.LGG=0 TO XN.LGG-1
50500   IF MIN.LGG>X.LGG(I.LGG,K.LGG) THEN MIN.LGG=X.LGG(I.LGG,K.LGG)
50510 NEXT
50520 RETURN
50530 *XLOGSC.LGG
50540 '
50550 '   LOG SCALE ( X-AXIS )
50560 '
50570 LINE(XMIN.LGG,YMAX.LGG)-(XMAX.LGG,YMAX.LGG)
50580 FOR I.LGG=XMIN.LGG TO XMAX.LGG
50590   FOR X.LGG=1 TO 10
50600     XSC.LGG=I.LGG+FNLOG10(X.LGG)             } 対数目盛の表示
50610     LINE(XSC.LGG,YMAX.LGG)-STEP(0,-.06)      }
50620   NEXT
50630   LINE(XSC.LGG,YMAX.LGG)-(XSC.LGG,YMIN.LGG),,,&H5555
```

```
50640 NEXT
50650 RETURN
50660 *YLOGSC.LGG
50670 '
50680 '  LOG SCALE ( Y-AXIS )
50690 '
50700 LINE(XMIN.LGG,YMAX.LGG)-(XMIN.LGG,YMIN.LGG)
50710 FOR I.LGG=YMIN.LGG TO YMAX.LGG
50720   FOR X.LGG=1 TO 10
50730     YSC.LGG=I.LGG-FNLOG10(X.LGG)
50740     LINE(XMIN.LGG,YSC.LGG)-STEP(.03,0)   } 対数目盛の表示
50750   NEXT
50760   LINE(XMAX.LGG,YSC.LGG)-(XMIN.LGG,YSC.LGG),,,&H9999
50770 NEXT
50780 RETURN
50790 *PLOT.LGG
50800 '
50810 '  PLOT DATA
50820 '
50830 FOR I.LGG=0 TO XN.LGG-1
50840   X.LGG=FNLOG10(X.LGG(I.LGG,0))
50850   Y.LGG=YMAX.LGG+YMIN.LGG-FNLOG10(X.LGG(I.LGG,1))
50860   CIRCLE (X.LGG,Y.LGG),.01,7
50870   PAINT (X.LGG,Y.LGG),7,7
50880 NEXT
50890 RETURN
50900 *PRMT.LGG
50910 '
50920 '   PRINT PARAMETER
50930 '
50940 CY.LGG=24
50950 XSC.LGG=XMAX.LGG-XMIN.LGG
50960 FOR I.LGG=0 TO XSC.LGG
50970   CX.LGG=INT(I.LGG*70/XSC.LGG)+4
50980   LOCATE CX.LGG,CY.LGG
50990   PRINT USING '1E+##';I.LGG+XMIN.LGG;
51000 NEXT
51010 CX.LGG=0
51020 YSC.LGG=YMAX.LGG-YMIN.LGG
51030 FOR I.LGG=0 TO YSC.LGG
51040   CY.LGG=INT(23-I.LGG*21/YSC.LGG)
51050   LOCATE CX.LGG,CY.LGG
51060   PRINT USING '1E+##';I.LGG+YMIN.LGG
51070 NEXT
51080 RETURN
51090 *TITLE.LGG
51100 PT.LGG=0
51110 *TITLE.LOOP.LGG
51120   IF (TITLE.LGG%(PT.LGG)=&HD0A) OR (PT.LGG=24) THEN RETURN
51130   PUT (32+PT.LGG*16,0),KANJI(TITLE.LGG%(PT.LGG))
51140   PT.LGG=PT.LGG+1
51150 GOTO *TITLE.LOOP.LGG
```

このグラフは，データの対数を取った値を表示しますから，当然0という値は存在しません．そのため，いわゆる原点（0，0）はなく，XとYの目盛の最小値が画面の左下隅みで重なります．よって，このグラフの目盛を決定するためには，データの最小値も求めなければなりません．また，両対数グラフはX，Y両方の値が，別のパラメータにより定まるということが多いので，このプログラムでは，どちらの値も配列に代入して引き渡されるようになっています．そこで，最小値，最大値を求める作業は，X，Y両方について行わなければなりません．さらに，これらの最大値，最小値も切りのいい値に直す訳ですが，ここでは，切りのいい値を 1×10^{n}（nは整数）としています．こうする事により，対数目盛のサイクルが途中でちぎれるということはなくなります．この計算を行っているのが*SCALEというサブルーチンです．

ここで計算された値により，あらかじめビューポートに指定された領域にワールド座標を割り当てます．このとき，WINDOW 文で指定される座標は実際の最大値，最小値でなく，その対数を取ったものです．

次に，目盛線を描きますが，これは，各サイクルに1から9までの対数を取った値の位置に目盛を描くだけです．これを，各軸のサイクル数だけ繰り返します．Y軸は，例によって引き算が必要です．

軸に振る数字は，1×10^nのところにのみです．キャラクタ座標の決定法は，リストの通りです．また，ここで表す数値はすべて指数表示にしてあります．

次に，データをプロットする座標について説明しましょう．ワールド座標は，すでにデータの対数を取った値がすべてグラフ内に収まるように設定されていますから，簡単にできます．x座標は，単にデータの値の対数を取るだけです．y座標については，例によって上下が逆転していますから，次のように補正します．

$$Y=2(YMIN+(YMAX-YMIN)/2)-\log_{10}X$$
$$=YMAX+YMIN-\log_{10}X$$

このプログラムでは，データの値に対応する位置に，小さな丸を描いていますが，もし折れ線を描きたいなら，座標の計算は上の様に行い，描れ線グラフの様に線を引けばいいでしょう．但し，この場合，XかYどちらかをソートしておかないと，線がメチャクチャになってしまいますから，気を付けてください．

☆サブルーチンの使い方

このサブルーチンを使うには，以下の変数，配列に必要なデータを代入して引き渡す必要があります（拡張子〝.LGG″は省略してあります）．

○XN
　データの数
○X(XN−1，1)
　X（n，0）：Xの値
　X（n，1）：Yの値
○TITLE%（24）
　タイトルの漢字コード

画面のハード・コピーを行うときはコピー・キーで行えますが，プログラム中で行いたい場合は，＊SCALEの直前のRETURN文の前か，メイン・ルーチンでこのサブルーチンを呼んだ後に行ってください．

5.3 漢字への応用

パーソナル・コンピュータをビジネスの分野で本格的に使おうとすると，どうしても漢字出力が必要になってきます．そこで，この章のしめくくりとして漢字を使ってビジネスの分野にアプローチしてみます．

PC-8801は，オプションの漢字 ROM を装備することにより，JIS 第1水準の約3000個の漢字の表示が可能となります．また，高解像度 CRT の使用により，一画面最大40×25行 (1000文字) の表示も可能です．この機能は，ちょっと前までのパーソナル・コンピュータでは，予想できなかった能力ですし，充分ビジネスに対応できるであろうことも予想できます．

5.3.1 漢字ワード・プロセッサ

ここでは，PC-8801にある漢字表示機能を生かした漢字ワード・プロセッサを作ります．

漢字ワード・プロセッサとは文書作製を能率よくするため，辞書検索や各種の編集機能を持ったエディタのことです．この節で作るワード・プロセッサも専用機にはかないませんが，基本的な辞書検索 (熟語は除く) や編集機能はありますし，最大254行，総文字数2500文字以上の能力を持っています．但し，ここで紹介するプログラムは，Disk-BASIC専用です．

☆データ構造

まず，このワード・プロセッサを理解して頂くために，そのデータ構造を説明しますが，グラフィックスとは直接的な関係はありませんから，プログラム解析に興味のない方は，漢字の表示の仕方だけ理解して，即実際の使い方を参照くださっても結構です．

では，一般のエディタのデータ構造の話から始めます．エディタの基本的な作業は，

1) 走査　　2) 変更　　3) 挿入　　4) 削除

の4つがあり，これらが高速にできなければ使い物になりません．当然のことですがメモリを大量に消費する構造も避けなければなりません．以上の仕様を満たす様にデータ構造を考えなければなりません．

まず考えられるのは，配列にベタ詰めに格納する方法です．この場合の各基本操作を考えますと，走査，変更は問題ありませんが，挿入と削除には問題あります．例えば，データ数を x としますと，データの移動量は平均 x／2になり，データが少ない場合は気になりませんが，データの量が増すにつれ時間がかかる様になってきます．従って，この方法はデータの少ない時にしか使えません．

では，挿入を簡単に，なおかつ高速に実現する方法はないのでしょうか，ここでリスト (プログラム・リストなどという狭い意味ではありません) という考え方を紹介します．これは，あるデータの次のデータの場所を示すポインタ用の配列を用意し，このポインタ用配列を次々と参照

することによって行う方法です．また，最後のデータのポインタには，これより後にデータはないという印（これを〝nil〟と表す）を入れます．先頭のデータへのリンク（リンクとはデータの結合のこと）は変数を１つ用意することによって実現します．

先頭への挿入は簡単に行えます．空いている所にデータを納めたらそのポインタを先頭へリンクし，先頭へのポインタをそのデータにリンクさせればいい訳です．この繰り返しより，リストはいくらでも長くできます．しかし，参照する順序は入力した順序の逆ですから，あまり望ましくありません．やはり入力した順に参照できる方が，より望ましく思えます．最後への追加も，先頭への挿入同様簡単にできますが，最後のデータを検索せねばなりません．そこで最後のデータを指すポインタを用意すればこの手間を省くことができます．

着目点へのポインタ
0
data
1
2
nil
←インデックス
←ポインタ

(a) 挿入，削除を行っていない時のデータのポインタのつながり方

着目点へのポインタ
0
1
2
nil
3

(b) 先頭への挿入

着目点へのポインタ
着目点へのポインタ
0
nil
nil

(c) 最後への追加

図３　リンクの方法(1)

図4 リンクの方法(2)

これで，リストはいくらでも長くすることができますし，途中への挿入もポインタの書き換えで容易にできます．

削除したい時は，削除したいデータをスキップする様に，ポインタを書き換えることによってこれを容易に実現できます．

このリスト構造の長所は，変更，挿入，削除が簡単にしかも高速にできることです．例えば，挿入は，データの代入，次のデータへのリンク，前のデータのポインタの書き換えの3ステップのみで実現できます．しかし，片方向リストのため順方向への走査は簡単ですが，逆方向への走査はできないため，先頭から走査し直さねばなりません．またデータ1つに対しポインタが1つ必要ですから，メモリの効率があまり良くありません．

逆方向への走査を可能にするためには，もう1つポインタ用配列を用意すればよいのですが，メモリ効率は増々悪くなります．しかし逆方向走査の魅力も捨てがたいものです．ところで1本のポインタで両方向を走査する方法があるのです．それが，逆転リンクとマジック・リストです．

逆転リンクは，順方向にリストを走査する時には，逆方向リストに，逆方向に走査する時には，順方向リストに書き換えながら走査するのです．実際には，現在参照中のデータを指すポインタの他に，その1つ前のデータを指すポインタを用意し，ポインタの更新の際にリストを逆につなぎ直すのです．

マジック・リストは，1つのポインタに正・逆両方向のデータを混合して持たせ，演算により分離して，新しいポインタを得る方法です．例を上げて説明すると，n 番目のデータへのポインタを A_nで表し，それを P という配列に格納するとします．P に格納する値は，

図5　逆転リンク

図6　マジック・リスト

$$P(A_n)=A_{n-1}\ \ \text{XOR (排他的論理和)}\ \ A_{n+1}$$

とします.

さて XORには,

$$A\ \ \text{XOR}\ \ B = C$$

としたときその定義より,

$$A = B\ \ \text{XOR}\ \ C$$

となります. このことを利用して,

$$A_{n+1} = P(A_n)\ \ \text{XOR}\ \ A_{n-1}$$
$$A_{n-1} = P(A_n)\ \ \text{XOR}\ \ A_{n+1}$$

という様にデータを分離することができます. この様にポインタを逆転リンクの様に 2 本持っていれば, 順方向にも逆方向にも走査できるのです.

この様なリスト構造は，エディタを設計する場合，必ず必要になってきますから，興味のある方は専門書を開いてみてください．

以上，基本的なデータ構造を説明しましたが，ここで，BASICによる漢字ワード・プロセッサには，どのデータ構造が向いているかを考えてみます．メモリ効率と速度とのバランスを考え，ベタ詰めのデータ構造と，リスト方式の両方の要所を取り入れるという方針で考えてみました．

メモリの効率と速度の向上を考え，とりあえず次のようなデータ構造にしてみます．

ポインタ	長さ	データ1	データ2		～

データ構造

●	25	
●	30	
●	40	
●	10	

上の行の長さが40個なのでこの2つは1続きのデータであることを示している

フラグの意味

図7　データ構造

データは，長さ40までの不定長とし，これを1つの単位としてリストを作ります．リストのリンクには，マジック・リストを採用します．データ長が40を越えたら，1行挿入して代入します．改行コードが入力されると40に満たなくても次へリンクします．データを挿入する際は，新しいリストを挿入し，データのコピーを作り挿入します．そして古いリストを削除します．あふれた分は次の行の先頭に挿入します．

この作業は，BASICが文字列演算をする際のメモリ操作とよく似ています．実際，ほぼ同じことをやることになります．それならいっそ，文字変数にデータを入れたらどうでしょう．文字変数なら可変長ですからメモリ効率はいいし，挿入，削除も関数を使えば簡単にできます．そして最大の利点として，外見上，固定長データとして扱えることです．このことは，リストの最大の弱点である，ごみの処理を大変簡単に実現してくれるのです．ここで言うごみとは，リスト処理中に発生するメモリ空間上の無駄な部分です．このごみは，放っておくとどんどん増え続け，メモリ空間を有効に使えなくなります．そこでガーベジ・コレクション（通称〝ちり集め〟）という作業をして，ごみを1箇所に集め，再利用できるようにする必要があるのです．この処理はBASICプログラムで行うと，膨大な手間と時間がかかるため，ちり集めをしなくてもいい様にする必要があった訳です．ところが，文字変数を利用すると，削除された部分をつないで空のリストを作るだけで，面倒な作業を大幅に省け，しかもBASIC内部では機械語で実行される訳ですから，

遙かに速くできる訳です．

以上の理由により，本ワード・プロセッサでは次の様なデータ構造に決定します．

テキストは，文字配列変数に入れる．各文字変数の最初から2文字はポインタ，3文字目以降をデータ領域とする．データは2 byte 1単位であるから，MKI$ 関数を使って変換した文字列で表現する．元に戻すにはCVI関数を使用する．リスト構造には，マジック・リストを採用する．もし1行のデータ数が40個を越えるときは，1行追加してデータを格納してゆく．データがない行は，あらかじめ単方向リストでリンクしておく．このリストをフリー・リストと呼ぶ．挿入が生じたときは，フリー・リストから消除し，消除した行を挿入にあてる．削除した行は，フリー・リストの先頭に追加してゆく．

1文字目	2文字目	3文字目	4文字目	
ポイ	ンタ	デー	タ	～

図8　データ構造(詳細版)

以上の様にデータ構造は決定しました．4章で紹介した機械語のパッケージの漢字 PUT は，このデータ構造に合わせて作られています．このパッケージでは，頭の2文字を無視して3文字目から2文字ずつを漢字コードと見なして画面に1行分 PUT します．パラメータの引き渡し方は，4．3．6の〝漢字1行 PUT〟及び本プログラムの system initialize を見てください．

system initialize の KP はストリング・ディスクリプタのアドレスを，KL は表示する行の指定を，WB は書き込む画面バンクをそれぞれ引き渡します．

☆漢字ワード・プロセッサの使い方

1．実行のさせ方

まず，機械語のパッケージがロードされているのを確認してください．もしロードせずに実行すると暴走します．この暴走はストップで止まるようなものではありませんので（リセットしか止まらない），必ずロードしてあるのを確認してください．

確認が済んだところで，本プログラムをロードして RUN します．しばらくすると，カーソルがホーム・ポジションに現れます．これでスクリーン・エディタが使用できます．

但し，このプログラムでは2つのファイルをオープンしますから，BASIC 起動時の "How many files" の問には，必ず" 2 "と答えてください．

また，このプログラムは，640×400モード用となっていますが，リストの16060行を変更することにより，640×200モード用としても使用できます．

2．スクリーン・エディタの使い方

英数字は，そのままタイプしてください．スクリーンに即表示されます．次の行へ進みたい時はリターン・キーをタイプしてください．カナ文字の入力は，カナ・シフトキーをロックしてタ

イプします．ひらがなを入れたい時は，さらにキャピタル・ロックします．但し，キャピタル・ロックを行うと，現在キー・バッファ中に溜っているカナ文字も，すべてひらがなと解釈されますから，先行入力は避けてください．

カーソル・キーを使えば，テキスト上の任意の場所にカーソルを持っていくことができます．もし，変更を加えたければ，既にある文字に重ねてタイプします．挿入したい時は，INS キーもしくは CTRL-R をタイプしてインサート・モードに入ってください．インサート・モードは，再び INS がタイプされるまで持続します．このモードでタイプすると，カーソルの左側に入力された文字が挿入されます．行と行の間に入たい時は，カーソルを挿入する行の直後の行の頭にセットして，CTRL-J をタイプすれば挿入されます．CTRL-J は分割するコマンドで，カーソルの右と左との一行を分割して 2 行にします．2 行を 1 行にしたい時は，前の行の最後へ行って DEL キーをタイプすれば，2 行が 1 行になります．一画面を越えた場合自動的にスクロールしますが，前の画面を見たい時は，HOME キー，もしくは CTRL-K をタイプします．逆に次の画面を見たい時は，CLR もしくは CTRL-L をタイプします．1 行スクロールは，ROLL UP, ROLL DOWN キーを使って行います．その他の機能は，別表を御覧ください．

ESC＋コントロール･キー	コントロール･キャラクタそのものを入力します．
ESC＋イ ESC＋エ	旧カナづかいの「ヰ」および「ゐ」，または「ヱ」および「ゑ」を入力します．
ESC＋数字＋コントロール･キャラクタ	数字の回数だけ，そのコマンドをくりかえします．
ESC＋数字＋ESC＋文字列＋ESC	数字の回数だけ，ESC間に挟まれた文字列をくりかえし入力します．文字はコントロール･キャラクタでもかまいません．
ESC＋数字＋英文配列＋スペース	英文字列の名前で数字の文字数だけ登録をします． 登録は100個まで可能です．
ESC＋英文配列＋スペース	英文字列の名前で登録済の文字を入力します．
ESC＋英文配列＋DEL	英文字列の名前での登録を抹消します．

表1　ESCシーケンスの一覧表

3．漢字辞書の使い方

スクリーン･エディタから辞書モードに移るには，CTRL-N をタイプします．辞書モードに入るとキーワードを聞いてきますから，引きたい漢字の読みをタイプしてください．キーワードが登録されている場合，対応する漢字を最下行に出力しますから，カーソルを動かして選びます．もし目的の漢字が出力されない時は，リターン・キーをタイプしてください．1行に表示しきれなかった場合は，その分の漢字のメニューが出力されます．目的の漢字がない場合は，ESC キーをタイプしてください．スクリーン・エディタに戻ります．目的の漢字が見つかった時は，その漢字のところにカーソルを合わせ，スペース・キーをタイプしてください．その漢字がスクリーン上に表示されます．

CTRL-A HELP	カーソルにある行に関する直前の変更を無効にする.
CTRL-B f・6	直前の語尾にカーソルを移動する.
CTRL-C STOP	入力可能な行数と文字数を表示する.
CTRL-D	ライン・エディタ・モードに入る.
CTRL-E	カーソルのある位置から行末までを消去する.
CTRL-F f・7	次の語の先頭にカーソルを移動する.
CTRL-G	カーソルのある位置よりその行の先頭までを消去する.
CTRL-H f・5	1つ前の文字を消去する.
CTRL-I TAB	次のタブ位置まで空白を挿入する.
CTRL-J f・3	カーソルの位置で行を分割する.
CTRL-K HOME	1画面前を表示する.
CTRL-L CLR	次の画面を表示する.
CTRL-M	次の行に改行する.
CTRL-N f・4	漢字入力モードに入る.
CTRL-O	何もしない.
CTRL-P	Qバッファの内容をカーソルのある行の前に挿入する.
CTRL-Q	タブをすべてリセットする.
CTRL-R INS	インサート・モード・スイッチを切りかえる.
CTRL-S	キー・バッファをクリアする.
CTRL-T	カーソルの位置にタブをセットする. 但し, すでにセットされているときはリセットする.
CTRL-U f・10	カーソルのある行を削除する.
CTRL-V f・1	半画面次を表示する.
CTRL-W f・2	半画面前を表示する.
CTRL-X f・9	カーソルを行末に移動する.
CTRL-Y	Qバッファにカーソルのある行を入力する.
CTRL-Z f・8	カーソルを行の先頭に移動する.
DEL	カーソルの下の文字を削除する. 但し行末の場合は, 次の行と合わせて1つ行にする.
→	カーソルを次の文字へ移動させる.
←	カーソルを前の文字に移動させる.
↑	カーソルを上のラインへ移動させる.
↓	カーソルを下のラインへ移動させる.

表2　スクリーン・エディタのコマンド一覧表

4．ライン・エディタ

ライン・エディタとは，スクリーン・エディタの様にカーソルを動かしてエディットする方式ではなく，テキストの行番号によりエディットするエディタです．

スクリーン・エディタからライン・エディタに移るには CTRL-D をタイプします．ライン・エディタでは，着目行に注意しないとテキストを破壊する恐れがあるので注意してください．

まず着目行の移動の仕方です．現在行を示す文字を〝.″(ピリオド)，最後の行を示す文字を〝$″(ダラー) で表します．〝.＋1″ で次の行を 〝.－1″ で直前の行を表せます．1は最初の行を表し，$－1は最後から2番目の行を表します．例えば，〝.＋1 〈CR〉″ とタイプすれば，次の行が表示され着目行が移ります．3行目から5行目を見たいときは，〝3，5 〈CR〉″ とタイプします．このとき着目行は3行目です．〝5，3〈CR〉″でも同じ表示が出ますが，このときの着目行は5行目です．

次にコマンドの説明をします．但し，αとβは行番号の指定，n は整数とします．

α〔，β〕〔=〕

着目行の移動およびテキストの表示．βの省略値は，α と同じ値です．〝=″ が付くと，α の値を表示します．

〔α〕〔，β〕L

αとβ の間のテキストをプリンタに出力．αの省略値は 1，βの省略値は 〝$″ です．

〔α〕〔，β〕W／ファイル・ネーム／〔A〕

αとβの間のテキストをファイルに出力します．Aのオプションがつくと既にあるファイルにアペンド (追加) されます．αの省略値は 1，βの省略値は 〝$″ です．着目行は$\alpha$ へ移動します．

〔α〕R／ファイル・ネーム／〔n〕

α の前にn行ファイルを読み込みます．αの省略値は〝.″(ピリオド)，nの省略値は32767です．着目行はαへ移動します．

〔α〕〔，β〕D

αからβまでを削除し，Qバッファに入れます．αの省略値は〝.″(ピリオド)，βの省略値は，αと同じ値です．着目行はαに移ります．

〔α〕P

αの前にQバッファを挿入します．α の省略値は 〝.″(ピリオド)，着目行はαに移ります．

〔α〕〔，β〕Y

αとβ の間のテキストをQバッファに読み込みます．αの省略値は〝.″(ピリオド)，βの省略値は，αと同じ値です．着目行はαに移ります．

〔α〕V　　スクリーン・エディタに戻ります．画面にはα行から１画面分表示されます．αの省略値は〝.〞（ピリオド）です．

Q　　本プログラムの実行を終了してコマンドレベルに戻ります．

以上で，漢字ワード・プロセッサの説明は終りですが，ここでこの漢字ワード・プロセッサの問題点を挙げておきます．大きな問題点として速度とキー入力が挙げられると思います．

まず速度の方ですが，表示は，機械語のライブラリを使用していますから，非常に高速です．挿入・削除などのテキスト操作も本格的なリスト構造を採用していますから，かなり実用になるものです．但し，文字列操作を頻繁に使用している関係から，ゴミが出やすく，最初に指摘したガーベジ・コレクションがかなりの確率で起こります（予想した以上に）．従って，何らかの処理の途中で，キー入力及びその他の動作を全く受け付けなくなることがあります．これは，まさしく BASIC がガーベジ・コレクションを行っている訳で，使用者はこの間待つ他ありません．

もう１つの問題点はキー入力で，PC-8801は BASIC が動いている間は，タイプ・アヘッド機能により，いかなる状況でも割り込みによりキー入力を受け付けます．しかし，このプログラムでは，専用の機械語ルーチンを使用していますから，機械語のルーチンを実行している間は BASIC の管理下をはなれます．従って，その間に入力されたデータは全く無視されてしまい，結果としてキー入力の取りこぼしとなってしまいます．

これら２つは，なかなか難しい問題です．特に後者は難しいでしょう．ガーベジ・コレクションの方は，なるべくゴミの出ない様な文字列処理を採用する，といったところでしょうか．

今回は，本来グラフィックという意味で紹介しましたので（多少本格的なものになってしまいましたが），熟語の検索や印字制御などの機能は貧弱です．このプログラムは，ほぼメモリの限界ですが，無駄なスペースを詰めて，マルチ・ステートメントを多用すれば，もう少し機能を増せるかもしれませんから挑戦してみてはいかがですか？

リスト7

```
10000 GOTO *MAIN メインルーチンへ
10010 '
10020 *NEXT. ポインタを１つ進める
10030   SWAP AP,BP : AP=CVI(T$[BP])XOR AP : RETURN
10040 '
10050 *BACK ポインタを１つ戻す
10060   AP=CVI(T$[BP])XOR AP : SWAP AP,BP : RETURN
10070 '
10080 *MOVE.LINE.TOP ポインタを行の先頭へ
10090   M=0 : IF AP=EOT OR BP=EOT THEN RETURN
10100 *MOVE.L.T0
10110   IF LEN(T$[BP])<MAXLEN THEN RETURN
10120   M=M+1 : GOSUB *BACK
10130   IF BP=EOT THEN RETURN ELSE *MOVE.L.T0
10140 '
10150 *INSERT ポインタの前に１行挿入する
10160   P=FR : FR=CVI(T$[FR])
10170   IF BP=EOT THEN *INSERT.0
10180    MID$(T$[BP],1,2)=MKI$((CVI(T$[BP])XOR AP)XOR P)
10190 *INSERT.0
10200   IF AP=EOT THEN *INSERT.1
10210    MID$(T$[AP],1,2)=MKI$((CVI(T$[AP])XOR BP)XOR P)
```

変数表

変数	内容
AP	着目点
BP	着目点の１つ前
FR	フリー・リストの先頭
T$(　)	テキスト
TB(　)	タブ
B$(　)	エスケープ登録用のバッファ
INS	インサート・モードスイッチ
DISP	画面表示フラグ
CX	カーソルxポジション
CY	カーソルyポジション

```
10220 *INSERT.1
10230   T$[P]=MKI$(AP XOR BP) : AP=P : RETURN
10240 '
10250 *SEVER テキストのリストからポインタを指すラインを切り離す
10260   P=CVI(T$[AP]) XOR BP : SWAP AP,P
10270   IF BP=EOT THEN *SEVER.0
10280   MID$(T$[BP],1,2)=MKI$((CVI(T$[BP])XOR P)XOR AP)
10290 *SEVER.0
10300   IF AP=EOT THEN RETURN
10310    MID$(T$[AP],1,2)=MKI$((CVI(T$[AP])XOR P)XOR BP)
10320   RETURN
10330 '
10340 *DELETE. テキストから切り離して，フリー・リストに戻す
10350   GOSUB *SEVER
10360   T$[P]=MKI$(FR) : FR=P : RETURN
10370 '
10380 *INS カーソルの前に1文字挿入する
10390   CC=CX+CX+3 : L=FALSE
10400   C$=LEFT$(T$[AP],CC-1)+MKI$(KC)
10410   T$[AP]=C$+RIGHT$(T$[AP],LEN(T$[AP])-CC+1)
10420 *INS.0
10430   GOSUB *NEXT.
10440   IF LEN(T$[BP])<=MAXLEN THEN *INS.1
10450    A$=LEFT$(T$[AP],2)+RIGHT$(T$[BP],LEN(T$[BP])-MAXLEN)
10460    T$[AP]=A$+RIGHT$(T$[AP],LEN(T$[AP])-2)
10470    T$[BP]=LEFT$(T$[BP],MAXLEN)
10480    GOSUB *INS.0
10490    GOTO *BACK
10500 *INS.1
10510   IF LEN(T$[BP])<MAXLEN THEN *INS.2
10520    L=TRUE : GOSUB *INSERT
10530 *INS.2
10540   GOTO *BACK
10550 '
10560 *INPCHAR 1文字挿入．INSが真ならば挿入する
10570   IF AP=EOT THEN GOSUB *INSERT
10580   CC=CX+CX+3 : IF UC=1 THEN GOSUB *INIT.U
10590   L=LEN(T$[AP])
10600   IF CC>L OR INS THEN GOSUB *INS ELSE MID$(T$[AP],CC,2)=MKI$(KC)
10610   IF DISP OR Q THEN *CSR.R
10620   IF NOT INS THEN GOSUB *PR.CHAR
10630   IF INS THEN IF L THEN GOSUB *DISP.UNDER ELSE GOSUB *DISP.LINE
10640   GOTO *CSR.R
10650 '
10660 *DEL カーソルの下の文字をテキストから削除
10670   L=2
10680 *DEL.0
10690   CC=CX+CX+3 : K=FALSE
10700   IF LEN(T$[AP])<MAXLEN THEN *DEL.1
10710    GOSUB *DEL0
10720    T$[AP]=T$[AP]+C$
10730 *DEL.1
10740   T$[AP]=LEFT$(T$[AP],CC-1)+RIGHT$(T$[AP],LEN(T$[AP])-CC-L+1)
10750   RETURN
10760 '
10770 *DEL0 長さが40を超えていたら，次の行の先頭を削除して後に付ける．
10780   GOSUB *NEXT.  これを再帰的に繰り返す．
10790   IF LEN(T$[AP])<MAXLEN THEN *DEL0.1
10800    GOSUB *DEL0
10810    T$[AP]=T$[AP]+C$
10820 *DEL0.0
10830    C$=MID$(T$[AP],3,L)
10840    T$[AP]=LEFT$(T$[AP],2)+RIGHT$(T$[AP],LEN(T$[AP])-L-2)
10850    GOTO *BACK
10860 *DEL0.1
10870   IF LEN(T$[AP])>L THEN *DEL0.0
10880   C$=MID$(T$[AP],3,L)
10890   K=TRUE : GOSUB *DELETE.
10900   GOTO *BACK
10910 '
10920 *CSR.R カーソルを次の文字へ
10930   CX=CX+1
10940 *PLUS.ADJ 行からあふれた時の補正
```

行からあふれた文字を次の行の先頭に挿入する．これをあふれなくなるまで，再帰的に繰り返す（10450〜10490行）

```
10950   IF AP=EOT THEN CX=0 : RETURN
10960   IF CX+CX+4<=LEN(T$[AP]) THEN RETURN
10970   IF LEN(T$[AP])=MAXLEN THEN *PLUS.0
10980   IF CX+CX+2<=LEN(T$[AP]) THEN RETURN
10990   CX=CX-1
11000 *PLUS.0
11010   CX=CX-LEN(T$[AP])¥2+1
11020   GOSUB *NEXT.
11030   CY=CY+1 : GOTO *PLUS.ADJ
11040 '
11050 *CSR.L カーソルを1つ戻す
11060   CX=CX-1
11070 *MINUS.ADJ 行からあふれた時の補正
11080   IF CX>=0 THEN RETURN
11090   IF BP=EOT THEN CX=0 : RETURN
11100   GOSUB *BACK
11110   IF LEN(T$[AP])<MAXLEN THEN CX=CX+1
11120   CX=CX+LEN(T$[AP])¥2-1
11130   CY=CY-1 : GOTO *MINUS.ADJ
11140 '
11150 *CSR.U カーソルを上の行へ
11160   IF BP=EOT THEN RETURN
11170   CY=CY-1 : GOSUB *BACK
11180 '
11190 *CX.ADJ カーソルが行からあふれていれば，その最後の文字に重ねる
11200   IF AP=EOT THEN CX=0 : RETURN
11210   IF CX+CX+2<=LEN(T$[AP]) THEN RETURN
11220   CX=LEN(T$[AP])¥2-1 : RETURN
11230 '
11240 *CSR.D カーソルを下の行へ
11250   IF AP=EOT THEN RETURN
11260   GOSUB *NEXT.
11270   CY=CY+1 : GOTO *CX.ADJ
11280 '
11290 *MOVE.CSR カーソルが画面からあふれているかどうかを調べて，再表示するかどうかを決める.
11300   OUT &H53,0
11310   IF CY<0 THEN CY=0 : DISP=TRUE
11320   IF CY>MAXLINE THEN CY=MAXLINE : DISP=TRUE
11330 *PR.CSR カーソルを表示する
11340   STOP OFF : CLS
11350   IF HI AND NOT INS THEN LOCATE CX,CY,1 : RETURN
11360   LOCATE ,,0 : COLOR 2
11370   IF HI THEN LOCATE CX,CY ELSE LOCATE CX,CY+CY+1
11380   IF INS EQV HI THEN PRINT "_"; ELSE PRINT "▬";
11390   RETURN
11400 '
11410 *JOIN 2行を1行にする
11420   IF AP=EOT THEN RETURN
11430   UC=1 : GOSUB *NEXT.
11440   IF AP=EOT THEN RETURN
11450   L=(MAXFIGURE-CX)*2+2
11460   T$[BP]=T$[BP]+SPACE$(L)
11470   GOSUB *BACK
11480   GOSUB *DEL.0
11490   GOTO *DISP.UNDER
11500 '
11510 *DELCHAR 1文字削除する．但し，行末の時は2行を1行にする.
11520   IF AP=EOT THEN RETURN
11530   IF CX+CX+3>LEN(T$[AP]) THEN *JOIN
11540   IF UC=1 THEN GOSUB *INIT.U
11550   GOSUB *DEL
11560   IF DISP OR Q  THEN RETURN
11570   IF K THEN *DISP.UNDER ELSE *DISP.LINE
11580 '
11590 *PRINT.LINE ポインタの指す行を表示する
11600   KL=CY : KP=VARPTR(T$[AP])
11610   DUMMY=USR(KPUT)
11620   KL=INSTR(3,T$[AP],CHR$(0))
11630   WHILE KL>0
11640    ZX=CX : ZY=CY
11650    CX=KL¥2-2 : KC=CVI(MID$(T$[AP],KL-1,2))
11660    GOSUB *PR.CHAR
11670    CX=ZX : CY=ZY
```

```
11680     KL=INSTR(KL+2,T$[AP],CHR$(0))
11690    WEND : RETURN
11700 '
11710 *SCROLL  SCだけスクロールする
11720    I=SC-CY : UC=1 : DISP=TRUE
11730    WHILE I>0 AND AP<>EOT
11740     I=I-1 : GOSUB *NEXT.
11750    WEND
11760    WHILE I<0 AND BP<>EOT
11770     I=I+1 : GOSUB *BACK
11780    WEND
11790    IF I>0 THEN CX=0 : CY=0 : RETURN
11800    IF I<0 THEN CY=0 : GOTO *CX.ADJ
11810    CY=CY+I : I=0
11820    WHILE I<CY AND AP<>EOT
11830     I=I+1 : GOSUB *NEXT.
11840    WEND
11850    GOTO *CX.ADJ
11860 '
11870 *DISP.UNDER カーソルから下を表示する
11880    P0=AP : P1=BP : J=CY
11890    WHILE CY<=MAXLINE AND AP<>EOT
11900     GOSUB *PRINT.LINE
11910     CY=CY+1 : GOSUB *NEXT.
11920    WEND
11930    IF CY>MAXLINE THEN *DISP.U0
11940     SCREEN ,1:LINE(0,CY*20)-(639,MAXLINE*20+16),0,BF:SCREEN ,0
11950 *DISP.U0
11960    AP=P0 : BP=P1 : CY=J : RETURN
11970 '
11980 *DISP.LINE カーソルのある行を表示する
11990    IF AP=EOT THEN RETURN
12000    P0=AP : P1=BP : J=CY
12010    WHILE CY<=MAXLINE AND LEN(T$[AP])=MAXLEN
12020     GOSUB *PRINT.LINE
12030     CY=CY+1 : GOSUB *NEXT.
12040    WEND
12050    IF CY<=MAXLINE THEN GOSUB *PRINT.LINE
12060    AP=P0 : BP=P1 : CY=J : RETURN
12070 '
12080 *DISP  1画面表示する
12090    IF NOT DISP THEN RETURN
12100    P0=AP : P1=BP : I=CY
12110    WHILE BP<>EOT AND CY>0 AND LOC(1)=0
12120     GOSUB *BACK
12130     CY=CY-1
12140    WEND
12150    WHILE AP<>EOT AND CY<0 AND LOC(1)=0
12160     GOSUB *NEXT.
12170     CY=CY+1
12180    WEND
12190    IF LOC(1)=0 THEN KP=0 : SCREEN,3 : DUMMY=USR(CLSC) : SCREEN,0
12200    WHILE AP<>EOT AND CY<=MAXLINE AND LOC(1)=0
12210     GOSUB *PRINT.LINE
12220     CY=CY+1 : GOSUB *NEXT.
12230    WEND
12240    IF AP=EOT OR CY>MAXLINE THEN DISP=FALSE
12250    IF NOT DISP THEN LINE(0,MAXLINE*20+17)-STEP(639,0),7,,&HFFF
12260    AP=P0 : BP=P1 : CY=I
12270    RETURN
12280 '
12290 *PR.CHAR  1文字表示する
12300    IF KC>32 THEN *PR.0
12310    PUT@(CX*16+8,CY*20),KANJI(KC+64),PSET
12320    KC=&H5E
12330 *PR.0
12340    PUT@(CX*16,CY*20),KANJI(KC),PSET
12350    RETURN
12360 '
12370 *CLR.Q.BUFF Qバッファをクリアする
12380    WHILE QP<>EOT
12390     P=QP : QP=ASC(T$[P])
12400     T$[P]=MKI$(FR) : FR=P
```

```
12410   WEND : RETURN
12420 '
12430 *INIT.U 復旧用にバック・アップをとる
12440   GOSUB *MOVE.LINE.TOP
12450   WHILE UP<>EOT : P=UP : UP=CVI(T$[P]) : T$[P]=MKI$(FR) : FR=P :
 WEND
12460   UC=0 : IF AP=EOT THEN RETURN
12470   WHILE LEN(T$[AP])=MAXLEN
12480    P=FR : FR=CVI(T$[FR])
12490    T$[P]=MKI$(UP)+RIGHT$(T$[AP],MAXLEN-2)
12500    UP=P : M=M-1 : GOSUB *NEXT.
12510   WEND
12520   P=FR : FR=ASC(T$[FR])
12530   T$[P]=MKI$(UP)+RIGHT$(T$[AP],LEN(T$[AP])-2) : UP=P
12540   WHILE M<0
12550    M=M+1 : GOSUB *BACK
12560   WEND : RETURN
12570 '
12580 *CTRL.A
12590   IF UC=1 THEN RETURN
12600   IF UC<>21 THEN GOSUB *CTRL.U0 ←── Uでデリートしたのでなければ,
12610   WHILE UP<>EOT                          その行を削除する.
12620    P=UP : UP=CVI(T$[P])
12630    IF BP=EOT THEN *HELP.1                                          ┐
12640     MID$(T$[BP],1,2)=MKI$((CVI(T$[BP])XOR AP)XOR P)                 │
12650 *HELP.1                                                           │
12660    IF AP=EOT THEN *HELP.2                                          │
12670     MID$(T$[AP],1,2)=MKI$((CVI(T$[AP])XOR BP)XOR P)                 │ バック・アップ
12680 *HELP.2                                                           │ を挿入する
12690    MID$(T$[P],1,2)=MKI$(BP XOR AP) : AP=P                          │
12700   WEND : UC=1                                                      │
12710   IF DISP THEN RETURN ELSE *DISP.UNDER                             ┘
12720 '
12730 *CTRL.B0
12740   GOSUB *CUR.L
12750 *CTRL.B
12760   IF AP=EOT THEN *CTRL.B1                                          ┐ 記号または
12770   IF CX=0 AND BP=EOT THEN RETURN                                   │ スペースまたは
12780   IF CX+CX+2=LEN(T$[AP]) THEN *CTRL.B1                             │ 改行文字を捜す
12790   IF CVI(MID$(T$[AP],CX+CX+3,2))>=&H2330 THEN *CTRL.B0             ┘
12800 *CTRL.B1
12810   GOSUB *CUR.L                                                     ┐
12820   IF AP=EOT THEN RETURN                                            │
12830   IF CX=0 AND BP=EOT THEN RETURN                                   │ 文字を捜す
12840   IF CX+CX+2=LEN(T$[AP]) THEN *CTRL.B1                             │
12850   IF CVI(MID$(T$[AP],CX+CX+3,2))<&H2330 THEN *CTRL.B1              ┘
12860   RETURN
12870 '
12880 *CTRL.C
12890   WIDTH 80 : LOCATE 0,CMNDLINE
12900   P=FR : I=0 : WHILE P<>EOT : P=CVI(T$[P]) : I=I+1 : WEND
12910   PRINT USING "free area is ### lines ";I,
12920   PRINT USING "#### bytes";FRE(0);
12930   FOR I=0 TO 1000 : NEXT
12940   WIDTH 40
12950   RETURN
12960 '
12970 *CTRL.D
12980   INS=FALSE : GOTO *LINE.EDITER
12990 '
13000 *CTRL.E
13010   IF UC=1 THEN GOSUB *INIT.U
13020   IF AP=EOT THEN RETURN
13030   IF LEN(T$[AP])<MAXLEN THEN *CTRL.E0
13040   GOSUB *NEXT.                  ┐
13050   WHILE LEN(T$[AP])=MAXLEN      │ 次の行に続いていればそれを削除
13060    DISP=TRUE : GOSUB *DELETE.   ┘
13070   WEND
13080   T$[AP]=LEFT$(T$[AP],2)
13090   GOSUB *BACK
13100 *CTRL.E0
13110   L=LEN(T$[AP])-CX-CX-2 : CC=CX+CX+3
13120   GOSUB *DEL.0
```

```
13130   IF NOT (DISP OR Q) THEN *DISP.LINE ELSE RETURN
13140 '
13150 *CTRL.F0
13160   GOSUB *CUR.R
13170 *CTRL.F
13180   IF AP=EOT THEN *CTRL.F1
13190   IF LEN(T$[AP])=CX+CX+2 THEN *CTRL.F1
13200   IF CVI(MID$(T$[AP],CX+CX+3,2))>=&H2330 THEN *CTRL.F0
13210 *CTRL.F1
13220   GOSUB *CUR.R
13230   IF AP=EOT THEN INS=FALSE : RETURN
13240   IF LEN(T$[AP])=CX+CX+2 THEN *CTRL.F1
13250   IF CVI(MID$(T$[AP],CX+CX+3,2))<&H2330 THEN *CTRL.F1
13260   RETURN
13270 '
13280 *CTRL.G
13290   IF UC=1 THEN GOSUB *INIT.U
13300   IF AP=EOT THEN RETURN
13310   GOSUB *MOVE.LINE.TOP
13320   CY=CY-M
13330   WHILE M>0                 ┐
13340    GOSUB *DELETE.           ├カーソル上の1行を削除する
13350    DISP=TRUE : M=M-1        ┘
13360   WEND
13370   L=CX*2 : CX=0
13380   IF L>0 THEN GOSUB *DEL.0
13390   IF NOT DISP THEN *PRINT.LINE ELSE RETURN
13400 '
13410 *CTRL.H
13420   IF CX>0 THEN CX=CX-1 : GOTO *DELCHAR
13430   IF AP=EOT THEN RETURN
13440   IF BP=EOT THEN *DELCHAR
13450   IF LEN(T$[BP])=MAXLEN THEN GOSUB *CSR.L
13460   GOTO *DELCHAR
13470 '
13480 *CTRL.I
13490   I=0 : J=0 : K=CX
13500   WHILE K<=MAXFIGURE AND I=0
13510    K=K+1 : J=J+1                                      ┐タブの位置を
13520    I!=2^(K AND 15) : IF I!>32767 THEN I!=I!-65536!   ┘捜す
13530    I=TB[K¥16] AND I!
13540   WEND
13550   KC=&H2120
13560   IF I=0 THEN RETURN
13570   IF UC=1 THEN GOSUB *INIT.U
13580   IF AP=EOT THEN GOSUB *INSERT
13590   FOR I=1 TO J
13600    GOSUB *INS
13610    GOSUB *CSR.R
13620   NEXT
13630   IF NOT DISP THEN IF L THEN GOSUB *DISP.UNDER ELSE GOSUB *DISP.
LINE
13640   RETURN
13650 '
13660 *CTRL.J  2行に分割する
13670   IF AP=EOT THEN *INSERT
13680   UC=1 : GOSUB *INSERT
13690   GOSUB *NEXT.
13700   T$[BP]=T$[BP]+MID$(T$[AP],3,CX+CX)
13710   GOSUB *BACK
13720   L=CX+CX : GOSUB *DEL0
13730   GOTO *DISP.UNDER
13740 '
13750 *CTRL.K
13760   SC=-1-MAXLINE : GOTO *SCROLL
13770 '
13780 *CTRL.L
13790   SC=MAXLINE+1 : GOTO *SCROLL
13800 '
13810 *CTRL.M
13820   IF AP=EOT THEN GOSUB *INSERT
13830   WHILE LEN(T$[AP])=MAXLEN
13840    CY=CY+1 : GOSUB *NEXT.
```

```
13850   WEND
13860   UC=1 : CX=0 : CY=CY+1 : GOTO *NEXT.
13870 '
13880 *CTRL.N
13890   GOTO *KDIC
13900 '
13910 *CTRL.O RETURN
13920 '
13930 *CTRL.P
13940   IF QP=EOT THEN RETURN
13950   UC=1 : GOSUB *MOVE.LINE.TOP
13960   CY=CY-M
13970   I=QP
13980   WHILE I<>EOT
13990    GOSUB *INSERT
14000    T$[AP]=LEFT$(T$[AP],2)+RIGHT$(T$[I],LEN(T$[I])-2)
14010    I=CVI(T$[I])
14020   WEND
14030   GOTO *DISP.UNDER
14040 '
14050 *CTRL.Q
14060   FOR I=0 TO MAXFIGURE¥16 : TB[I]=0 : NEXT
14070   RETURN
14080 '
14090 *CTRL.R
14100   INS=NOT INS : RETURN
14110 '
14120 *CTRL.S RETURN
14130 '
14140 *CTRL.T
14150   I!=2^(CX AND 15) : IF I!>32767 THEN I!=I!-65536!
14160   TB[CX¥16]=TB[CX¥16] XOR I!
14170   RETURN
14180 '
14190 *CTRL.U
14200   IF AP=EOT THEN RETURN ELSE GOSUB *CTRL.U0
14210   CX=0 : UC=21 : GOTO *DISP.UNDER
14220 *CTRL.U0
14230   GOSUB *MOVE.LINE.TOP
14240   CY=CY-M : IF UC=1 THEN GOSUB *INIT.U
14250   WHILE LEN(T$[AP])=MAXLEN
14260    GOSUB *DELETE.
14270   WEND
14280   GOTO *DELETE.
14290 '
14300 *CTRL.V
14310   SC=MAXLINE¥2+1 : GOSUB *SCROLL
14320 *MOVE.CENTER
14330   I=MAXLINE¥2-CY
14340   WHILE I>0
14350    I=I-1 : GOSUB *CSR.D
14360   WEND
14370   WHILE I<0
14380    I=I+1 : GOSUB *CSR.U
14390   WEND
14400   RETURN
14410 '
14420 *CTRL.W
14430   SC=-MAXLINE¥2-1 : GOSUB *SCROLL
14440   GOTO *MOVE.CENTER
14450 '
14460 *CTRL.X
14470   IF AP=EOT THEN RETURN
14480   WHILE LEN(T$[AP])=MAXLEN
14490    GOSUB *CSR.D
14500   WEND
14510   CX=LEN(T$[AP])¥2-1
14520   RETURN
14530 '
14540 *CTRL.Y
14550   IF AP=EOT THEN RETURN ELSE M=0
14560   GOSUB *CLR.Q.BUFF
```

```
14570    GOSUB *YANK.Q.BUFF
14580    WHILE M>0
14590     M=M-1 : GOSUB *NEXT.
14600    WEND
14610    WHILE M<0
14620     M=M+1 : GOSUB *BACK
14630    WEND
14640    RETURN
14650 '
14660 *YANK.Q.BUFF
14670    P=FR : FR=CVI(T$[FR])
14680    WHILE LEN(T$[AP])=MAXLEN
14690     T$[P]=T$[AP] : MID$(T$[P],1,2)=MKI$(QP)
14700     M=M-1 : QP=P : P=FR : FR=CVI(T$[FR]) : GOSUB *NEXT.
14710    WEND
14720    T$[P]=T$[AP] : MID$(T$[P],1,2)=MKI$(QP) : QP=P
14730    RETURN
14740 '
14750 *CTRL.Z
14760    IF AP=EOT THEN RETURN
14770    GOSUB *MOVE.LINE.TOP
14780    CY=CY-M : CX=0 : RETURN
14790 '
14800 *ESC
14810    C=0 : X=ASC(INPUT$(1,1))
14820    WHILE X>=48 AND X<58
14830     C=C*10+X-48
14840     X=ASC(INPUT$(1,1))
14850    WEND
14860    IF (X>64 AND X<127) OR C=0 THEN *ESC1 ELSE E$=CHR$(X)
14870    IF X<>27 THEN X=27 ELSE E$='' : X=ASC(INPUT$(1,1))
14880    WHILE X<>27
14890     E$=E$+CHR$(X) : X=ASC(INPUT$(1,1))
14900    WEND
14910    IF E$='' THEN RETURN
14920    FOR II=1 TO C
14930     FOR III=1 TO LEN(E$)
14940      CH=ASC(MID$(E$,III,1))
14950      GOSUB *EDIT.0
14960     NEXT
14970    NEXT
14980    RETURN
14990 *ESC1
15000    J=0 : A$=''
15010    WHILE X>=48 AND X<127
15020     A$=A$+CHR$(X) : J=J+1
15030     X=ASC(INPUT$(1,1))
15040    WEND
15050    IF C=0 THEN *ESC3
15060    I=0 : J=JJ : K=0                                  ← 登録にあるか
15070 *REPEAT0                                             ← どうか捜す
15080     K=(I+J)¥2 : C$=LEFT$(B$[K],INSTR(B$[K],' '))
15090     IF A$>C$ THEN I=K+1 ELSE J=K-1
15100    IF A$<>C$ AND I<=J THEN *REPEAT0
15110    IF A$=C$ THEN *ESC2
15120    IF A$>C$ THEN K=K+1
15130    IF JJ=MAXBUF THEN FOR I=0 TO 40: BEEP 1: BEEP 0: NEXT: RETURN
15140    FOR I=JJ TO K STEP -1
15150     SWAP B$[I],B$[I+1]          ← 新規登録用のスペースを空ける
15160    NEXT
15170    JJ=JJ+1
15180 *ESC2
15190    B$[K]=A$+' '
15200    I=1 : OUT &H53,1
15210    WHILE I<=C AND AP<>EOT AND LEN(B$[K])<254         ← 登録を行う.
15220     CC=CX+CX+3                                       ← 行末文字は,$0D0A
15230     IF LEN(T$[AP])<CC THEN KC=&HD0A ELSE KC=0        ← とする
15240     IF KC=0 THEN KC=CVI(MID$(T$[AP],CC,2))
15250     B$[K]=B$[K]+MKI$(KC)
15260     GOSUB *CSR.R
15270     I=I+1
15280    WEND
15290    OUT &H53,0 ←── キャラクタ画面OFF
```

```
15300   RETURN
15310 *ESC3
15320   IF LEN(A$)>0 THEN *ESC4 ELSE KC=X
15330   IF X=&HB2 THEN KC=&H2570 : GOSUB *CAPS.CHECK
15340   IF X=&HB4 THEN KC=&H2571 : GOSUB *CAPS.CHECK   エスケープ・キャラクタ
15350   IF X=&HB6 THEN KC=&H2575                        を入れる
15360   IF X=&HB9 THEN KC=&H2576
15370   IF NOT(X=13 OR X=10) THEN *INPCHAR
15380   CH=X : GOSUB *CTRL.J
15390   GOTO *CSR.R
15400 *ESC4
15410   I=0 : J=JJ
15420   WHILE I<=J
15430    K=(I+J)¥2 : L=INSTR(B$[K]," ") : C$=LEFT$(B$[K],L-1)
15440    IF A$=C$ THEN I=J+1 ELSE IF A$>C$ THEN I=K+1 ELSE J=K-1
15450   WEND
15460   IF A$<>C$ THEN FOR I=0 TO 40:BEEP 1:BEEP 0:NEXT:RETURN
15470   IF X<>&H7F AND X>31 THEN *ESC5
15480   FOR I=K TO JJ-1
15490    SWAP B$[I],B$[I+1]
15500   NEXT                      登録の抹消
15510   JJ=JJ-1
15520   RETURN
15530 *ESC5
15540   LL=L+1 : Q=TRUE
15550   OUT &H53,1 ←── キャラクタ画面ON
15560   WHILE LL<LEN(B$[K])
15570    KC=CVI(MID$(B$[K],LL,2))
15580    IF KC=&HD0A THEN GOSUB *CTRL.M ELSE GOSUB *INPCHAR
15590    IF CY>MAXLINE THEN DISP=TRUE                       登録文字の入力
15600    IF DISP OR CX>0 THEN *ESC6
15610     GOSUB *CSR.L
15620     GOSUB *PRINT.LINE
15630     GOSUB *CSR.R
15640 *ESC6
15650    LL=LL+2
15660   WEND
15670   GOSUB *PRINT.LINE
15680   OUT &H53,0 : Q=FALSE
15690   RETURN
15700 '
15710 *CUR.R
15720   IF AP=EOT THEN RETURN
15730   IF CX+CX+4=LEN(T$[AP]) THEN UC=1
15740   GOTO *CSR.R
15750 '
15760 *CUR.L
15770   IF CX=0 AND BP=EOT THEN RETURN
15780   IF CX>0 THEN *CSR.L
15790   IF CX=0 AND LEN(T$[BP])<MAXLEN THEN UC=1
15800   GOTO *CSR.L                                行を移ったかどうか
15810 '                                           チェックしながら
15820 *CUR.U                                       カーソルを動かす
15830   IF BP=EOT THEN RETURN
15840   IF LEN(T$[BP])<MAXLEN THEN UC=1
15850   GOTO *CSR.U
15860 '
15870 *CUR.D
15880   IF AP=EOT THEN RETURN
15890   IF LEN(T$[AP])<MAXLEN THEN UC=1
15900   GOTO *CSR.D
15910 '
15920 *ROLUP
15930   SC=1 : GOTO *SCROLL
15940 '
15950 *ROLDWN
15960   SC=-1 : GOTO *SCROLL
15970 '
15980 *MAIN
15990   CLEAR ,&HE3FF
16000   DEFINT A-Z
16010   OPEN "kybd:" FOR INPUT AS #1
```

```
16020 '            →これをFALSEにすると640×200のモードとなる
16030   TRUE=(1=1) : FALSE=NOT TRUE
16040   MAXFIGURE=39 : IF MAXFIGURE>39 THEN MAXFIGURE=39
16050   MAXLEN=MAXFIGURE*2+4
16060   HI=TRUE  : IF HI THEN MAXLINE=17 ELSE MAXLINE=8
16070   IF HI THEN CMNDLINE=MAXLINE+1 ELSE CMNDLINE=MAXLINE*2+2
16080   MAXTEXT=299 : EOT=MAXTEXT+1 : MAXBUF=100
16090   DIM T$[MAXTEXT],B$[MAXBUF],TB[MAXFIGURE¥16]
16100   DIM KY$[23],KCD[23],CNT[23]
16110 '*      →最大行数の決定
16120 '*    initialize graphic package
16130 '*
16140   DEF USR=&HE400
16150   PARM=&HE5F8
16160   KP=0 : KL=0 : WB=0 : DUMMY = 0
16170   W=0 : PW=VARPTR(W)
16180   W=VARPTR(KP) : POKE PARM,  PEEK(PW) : POKE PARM+1,PEEK(PW+1)
16190   W=VARPTR(KL) : POKE PARM+2,PEEK(PW) : POKE PARM+3,PEEK(PW+1)
16200   W=VARPTR(WB) : POKE PARM+4,PEEK(PW) : POKE PARM+5,PEEK(PW+1)
16210 '
16220 *COLD
16230   FOR I=1 TO 10
16240    KEY I,CHR$(ASC(MID$('VWJNHBFZXU',I))-64) ←ファンクション・キーの定義
16250   NEXT
16260   FOR I=0 TO MAXTEXT : T$[I]=MKI$(I+1) : NEXT ←フリー・リストをつなぐ
16270   FOR I=0 TO MAXFIGURE¥16 : TB[I]=&H8080 : NEXT ←タグの初期化
16280   B$[0]=MKI$(&H20FF) : JJ=0
16290   AP=EOT : BP=EOT : FR=0 : LPT=FALSE : QP=EOT : UP=EOT
16300 '
16310 *HOT
16320   CX=0 : CY=0 : ON STOP GOSUB *CTRL.C
16330   IF HI THEN KPUT=17 : CLSC=2 : SCREEN 2,0
16340   IF NOT HI THEN KPUT=16 : CLSC=1 : WB=0 : SCREEN 1,0,0,1
16350   WIDTH 40,20 : CONSOLE 0,25,0,0
16360   ON ERROR GOTO *TRAP
16370   SCREEN ,3 : KP=0 : DUMMY=USR(CLSC) : SCREEN ,0
16380   Q=FALSE : DISP=TRUE : KC=0 : CH=0  : UC=1
16390 '*
16400 '*    screen editing mode
16410 '*
16420 *EDIT.
16430   C0=CH
16440   IF LOC(1)=0 THEN GOSUB *MOVE.CSR キーバッファにデータが留っていれば
16450   IF LOC(1)=0 THEN GOSUB *DISP     表示を止めてその処理に移る
16460   STOP ON : CH=ASC(INPUT$(1,1))
16470   GOSUB *EDIT.0
16480   GOTO *EDIT.
16490 *EDIT.0
16500   STOP OFF
16510   IF CH<32 OR CH=127 OR CH>247 THEN *CTRL
16520   IF CH<&HDE THEN *EDIT.3
16530   IF CH=&HDF THEN *EDIT.1
16540 '
16550   KC=&H212B
16560   IF C0=&HB3 AND (INP(10)AND 128)<>0 THEN I=&H4E : GOTO *EDIT.2
16570   IF C0<&HB6 THEN *INPCHAR
16580   IF C0>&HC4 AND C0<&HCA THEN *INPCHAR  ダク音の補正
16590   IF C0>&HCE THEN *INPCHAR
16600   I=1
16610   GOTO *EDIT.2
16620 '
16630 *EDIT.1
16640   KC=&H212C                              }
16650   IF C0<&HCA OR C0>&HCE THEN *INPCHAR    }半ダク音の補正
16660   I=2                                    }
16670 *EDIT.2
16680   GOSUB *CSR.L
16690   KC=CVI(MID$(T$[AP],CX+CX+3,2))+I
16700   MID$(T$[AP],CX+CX+3,2)=MKI$(KC)
16710   IF NOT DISP THEN GOSUB *PR.CHAR
16720   GOSUB *CSR.R
16730   RETURN
16740 '
```

```
16750 *EDIT.3
16760   GOSUB *TRKANJI
16770   GOTO *INPCHAR
16780 '
16790 *TRKANJI JISコードを変字コードに変換する
16800   KC=0 : I=CH
16810   IF CH>128 THEN *KANA
16820 '
16830   IF I>47 AND I<58 THEN KC=I+&H2300
16840   IF I>64 AND I<91 THEN KC=I+&H2300
16850   IF I>96 AND I<123 THEN KC=I+&H2300
16860   IF KC<>0 THEN RETURN
16870   RESTORE *TRDATA
16880   WHILE I>31 : READ TD$ : I=I-32 : WEND
16890   KC=ASC(MID$(TD$,I+1,1))+&H2100
16900   RETURN
16910 '
16920 *TRDATA
16930   DATA "  *mtpsulJKv¥$]%?              '(cad)"
16940   DATA "w                               No002"
16950   DATA "                                PBQA "
16960 '
16970 *KANA
16980   IF I>159 AND I<166 THEN *KSPECIAL
16990   IF I=176 THEN KC=&H213D : RETURN
17000 '
17010   RESTORE *KTRDATA
17020   READ TD$
17030   WHILE I>&HBF : I=I-32 : READ TD$ : WEND
17040   KC=ASC(MID$(TD$,I-159,1))+&H2500
17050 *CAPS.CHECK
17060   IF (INP(10) AND 128)=0 THEN KC=KC-&H100
17070   RETURN
17080 '
17090 *KTRDATA
17100   DATA "         r!#%')cegC 「$&(*+-/13579;="
17110   DATA "?ADFHJKLMNORUX[^_=abdfhijklmos      "
17120 '
17130 *KSPECIAL
17140   KC=ASC(MID$(" 3fg26",CH-&H9F,1))+&H20F0
17150   RETURN
17160 '
17170 *CTRL
17180   IF CH<127 THEN *CTRL0
17190   IF CH=127 THEN *DELCHAR
17200   IF CH=248 THEN *ROLUP
17210   IF CH=249 THEN *ROLDWN
17220 *CTRL0
17230   ON CH    GOTO *CTRL.A,*CTRL.B,*CTRL.C,*CTRL.D,*CTRL.E
17240   ON CH- 5 GOTO *CTRL.F,*CTRL.G,*CTRL.H,*CTRL.I,*CTRL.J
17250   ON CH-10 GOTO *CTRL.K,*CTRL.L,*CTRL.M,*CTRL.N,*CTRL.O
17260   ON CH-15 GOTO *CTRL.P,*CTRL.Q,*CTRL.R,*CTRL.S,*CTRL.T
17270   ON CH-20 GOTO *CTRL.U,*CTRL.V,*CTRL.W,*CTRL.X,*CTRL.Y
17280   ON CH-25 GOTO *CTRL.Z,*ESC,*CUR.R,*CUR.L,*CUR.U,*CUR.D
17290   RETURN
17300 '
17310 *KDIC 漢字辞書
17320   P0=CX : P1=CY : CY=MAXLINE+1
17330   RESTORE *KYWORD
17340   FOR I=0 TO 23 : READ KY$[I],KCD[I],CNT[I] : NEXT
17350   COLOR 0 : WIDTH 80 : LOCATE 0,CMNDLINE,1
17360   CONSOLE CMNDLINE,1
17370   INPUT "ｶﾝｼﾞ ﾉ ﾖﾐ ";A$
17380   L=0 : R=23 : K=(L+R)¥2
17390  *REPEAT
17400    K=(L+R)¥2
17410     IF A$>KY$[K] THEN L=K+1 ELSE R=K-1
17420   IF L<=R AND A$<>KY$[K] THEN *REPEAT
17430   IF A$<KY$[K] THEN K=K-1 : IF K<0 THEN QUIT=TRUE : GOTO *KDIC2
17440   IF A$=KY$[K] THEN *KDIC1 ELSE KCD[0]=KCD[K]
17450   GOSUB *KDIC0
17460   FOR I=1 TO 23 : READ KY$[I],KCD[I],CNT[I] : NEXT
17470   FOR I=1 TO 23 : KCD[I]=KCD[I-1]+KCD[I] : NEXT
```

```
17480   L=1 : R=23
17490  *REPEAT1
17500    K=(L+R)¥2
17510    IF A$>KY$[K] THEN L=K+1 ELSE R=K-1
17520   IF L<=R AND A$<>KY$[K] THEN *REPEAT1
17530   IF A$<>KY$[K] THEN QUIT=TRUE : GOTO *KDIC2
17540 *KDIC1
17550   KCD=KCD[K] : CNT=CNT[K]
17560   WIDTH 40
17570   I=0 : QUIT=FALSE : FIND=FALSE
17580   WHILE NOT FIND AND NOT QUIT
17590    J=0 : KC=KCD-1
17600    FOR K=0 TO I
17610     KC=KC+1
17620     X=(KC+129)AND 255
17630     IF X<162 THEN KC=KC-X+162
17640    NEXT
17650    K=KC
17660    IF CNT-I-1>MAXFIGURE THEN J=MAXFIGURE ELSE J=CNT-I-1
17670    FOR L=0 TO J
17680     X=(K+129)AND 255
17690     IF X<162 THEN K=K-X+162
17700     PUT(L*16,MAXLINE*20+20),KANJI(K),PSET
17710     K=K+1
17720    NEXT
17730    IF J=MAXFIGURE THEN *KDIC3
17740    LINE(J*16+16,MAXLINE*20+20)-(639,MAXLINE*20+36),0,BF
17750 *KDIC3
17760    CX=0 : NX=FALSE
17770    WHILE NOT (FIND OR QUIT OR NX)
17780     GOSUB *PR.CSR
17790     A=ASC(INPUT$(1,1))
17800     IF A=32 THEN FIND=TRUE : A=0
17810     IF A=27 THEN QUIT=TRUE : A=0
17820     IF A=28 THEN CX=CX+1 : A=0 : IF CX>J THEN CX=0
17830     IF A=29 THEN CX=CX-1 : A=0 : IF CX<0 THEN CX=J
17840     IF A<>0 THEN NX=TRUE
17850    WEND
17860    IF NX THEN IF J=MAXFIGURE THEN I=I+J+1 ELSE I=0
17870    IF I>=CNT THEN I=0
17880   WEND
17890   IF QUIT THEN *KDIC2
17900   FOR K=1 TO CX
17910    KC=KC+1 : X=(KC+129)AND 255
17920    IF X<162 THEN KC=KC-X+162
17930   NEXT
17940 *KDIC2
17950   WIDTH 40 : LINE(0,MAXLINE*20+20)-STEP(639,20),0,BF
17960   CX=P0 : CY=P1
17970   IF QUIT THEN RETURN ELSE *INPCHAR
17980
17990 *KDIC0
18000   ON K+1 GOTO *K0,*K1,*K2,*K3,*K4,*K5,*K6,*K7,*K8,*K9
18010   ON K-9 GOTO *K10,*K11,*K12,*K13,*K14,*K15,*K16,*K17
18020   ON K-17 GOTO *K18,*K19,*K20,*K21,*K22,*K23
18030
18040 *LINE.EDITER
18050   GOSUB *MOVE.LINE.TOP
18060 *LINE.EDIT0
18070   LC=1 : P0=AP : P1=BP : INS=FALSE : STOP OFF
18080   WHILE BP<>EOT
18090    GOSUB *BACK
18100    LC=LC+1 : GOSUB *MOVE.LINE.TOP
18110   WEND
18120   AP=P0 : BP=P1                          総行数のカウント
18130   LAST=LC
18140   WHILE AP<>EOT
18150    GOSUB *CTRL.M
18160    LAST=LAST+1
18170   WEND
18180   AP=P0 : BP=P1 : CY=MAXLINE
18190   WIDTH 80
18200   CONSOLE CMNDLINE,20-CMNDLINE
```

```
18210   CLS : LOCATE ,,1
18220   COLOR 0 : OUT &H53,0
18230   LINE(639,MAXLINE*20+17)-STEP(-639,0),0
18240   LINE-STEP(639,0),7,,&HC3F0
18250 '
18260 *LINE.EDIT
18270   LINE INPUT CM$
18280   IF LEN(CM$)=0 THEN *LINE.EDIT ELSE A$=LEFT$(CM$,1) : CC=1
18290   WHILE A$=" " : CC=CC+1 : A$=MID$(CM$,CC,1) : WEND
18300   IF INSTR(".$+-0123456789",A$)=0 THEN SL=0 : GOTO *LINE.0
18310   GOSUB *EXPRESSION
18320   SL=VALUE
18330   IF SL<1 THEN SL=1
18340   IF SL>LAST THEN SL=LAST
18350 *LINE.0
18360   IF A$<>"," THEN EL=0 : GOTO *LINE.1
18370   WHILE A$=" " : CC=CC+1 : A$=MID$(CM$,CC,1) : WEND
18380   CC=CC+1 : A$=MID$(CM$,CC,1)
18390   IF INSTR(CHR$(0)+".$+-0123456789",A$)<2 THEN *LINE.EDIT
18400   GOSUB *EXPRESSION
18410   EL=VALUE
18420   IF EL<1 THEN EL=1
18430   IF EL>LAST THEN EL=LAST
18440 *LINE.1
18450   GOSUB *LINE.2
18460   GOTO *LINE.EDIT
18470 *LINE.2
18480   CM=(INSTR(MKI$(0)+"==LlRrWwVvQqDdYyPp",A$)+1)¥2
18490   IF CM=0 THEN RETURN
18500   ON CM GOTO *SCRN.PRINT,*LINE.NUM,*LPT.PRINT,*READ.FILE
18510   ON CM-4 GOTO *WRITE.FILE,*VIDEO,*QUIT,*DELETE.LINE,*YANK,*PUT.
18520 '
18530 *QUIT
18540   PRINT "Sure ? (y/n)";
18550   A$=INPUT$(1,1) : PRINT A$
18560   IF A$<>"Y" AND A$<>"y" THEN RETURN
18570   DUMMY=USR(0)
18580   IF LPT THEN LPRINT CHR$(27)+"H";
18590   COLOR 0 : SCREEN 0,0 : WIDTH 80,25
18600   CONSOLE 0,25,0,0
18610   END
18620 '
18630 *EXPRESSION  行の指定を評価する
18640   SIGN=FALSE : VALUE=0
18650 *EXPR
18660   WHILE A$=" " : CC=CC+1 : A$=MID$(CM$,CC,1) : WEND
18670   IF A$="+" THEN SIGN=FALSE : CC=CC+1 : A$=MID$(CM$,CC,1)
18680   IF A$="-" THEN SIGN=TRUE : CC=CC+1 : A$=MID$(CM$,CC,1)
18690   IF A$="." THEN NUM=LC:CC=CC+1:A$=MID$(CM$,CC,1):GOTO *EXPR1
18700   IF A$="$" THEN NUM=LAST:CC=CC+1:A$=MID$(CM$,CC,1):GOTO *EXPR1
18710   NUM=0 : WHILE A$=" " : CC=CC+1 : A$=MID$(CM$,CC,1) : WEND
18720   WHILE A$>="0" AND A$<="9"
18730    NUM=NUM*10+ASC(A$)-48
18740    CC = CC + 1 : A$=MID$(CM$,CC,1)
18750   WEND
18760 *EXPR1
18770   IF SIGN THEN NUM=-NUM
18780   VALUE=VALUE+NUM
18790   WHILE A$=" " : CC=CC+1 : A$=MID$(CM$,CC,1) : WEND
18800   IF A$="+" OR A$="-" THEN *EXPR
18810   RETURN
18820 '
18830 *MOVE.LINE  SLの示す行へ移る
18840   WHILE SL<LC
18850    GOSUB *BACK
18860    GOSUB *MOVE.LINE.TOP
18870    LC=LC-1
18880   WEND
18890   WHILE SL>LC
18900    WHILE LEN(T$[AP])=MAXLEN
18910     GOSUB *NEXT.
18920    WEND
18930    LC=LC+1 : GOSUB *NEXT.
```

```
18940    WEND
18950    RETURN
18960 '
18970 *LINE.NUM   行番号を表示する
18980    NUM=TRUE : GOTO *SCRN.0
18990 '
19000 *SCRN.PRINT  行番号を表示しない
19010    NUM=FALSE
19020 *SCRN.0
19030    IF SL=0 THEN RETURN
19040    IF EL=0 THEN EL=SL
19050    I=SL
19060    IF SL>EL THEN SWAP SL,EL
19070    GOSUB *MOVE.LINE
19080    ON STOP GOSUB *ESCAPE
19090    Q=TRUE : STOP ON
19100    WHILE LC<=EL AND AP<>EOT AND Q
19110     WHILE LEN(T$[AP])=MAXLEN
19120      SCREEN ,1 : ROLL 20 : SCREEN ,0
19130      GOSUB *PRINT.LINE
19140      GOSUB *NEXT.
19150     WEND
19160     SCREEN ,1 : ROLL 20 : SCREEN ,0
19170     GOSUB *PRINT.LINE
19180     GOSUB *NEXT.
19190     LC=LC+1
19200    WEND
19210    IF EL=LAST AND Q THEN FOR J=0 TO 40 : BEEP 1 : BEEP 0 : NEXT
19220     LINE(0,MAXLINE*20+17)-STEP(639,0),7,,&HC3F0
19230    SL=I : Q=FALSE
19240    GOSUB *MOVE.LINE
19250    IF NUM THEN PRINT LC
19260    RETURN
19270 '
19280 *ESCAPE
19290    STOP OFF : Q=FALSE : RETURN
19300 '
19310 *LPT.PRINT
19320    IF SL=0 THEN SL=1
19330    IF EL=0 THEN EL=LAST
19340    OPEN "lpt:" FOR OUTPUT AS #2
19350    IF NOT LPT THEN LPT=TRUE : PRINT #2, CHR$(27)+"K";
19360    L=0 : R=32767 : WF=FALSE
19370 *PRINT.FILE
19380    I=SL
19390    IF SL>EL THEN SWAP SL,EL
19400    GOSUB *MOVE.LINE
19410    WHILE LC<=EL AND AP<>EOT
19420     K=0 : Q=FALSE
19430     WHILE LEN(T$[AP])=MAXLEN
19440      Q=TRUE
19450      FOR J=3 TO MAXLEN-1 STEP 2
19460       WHILE K<L : PRINT #2,MKI$(&H2021); : K=K+1 : WEND
19470       PRINT #2, MID$(T$[AP],J+1,1); MID$(T$[AP],J,1);
19480       K=K+1 : IF K>=R THEN K=0 : PRINT #2,
19490      NEXT
19500      GOSUB *NEXT.
19510     WEND
19520     IF Q OR WF OR LEN(T$[AP])<3 THEN *LPT.0
19530     IF CVI(MID$(T$[AP],3,2))<>27 THEN *LPT.0           ┐
19540     J=5 : Y=TRUE : L=0 : R=0                           │
19550     WHILE J<LEN(T$[AP]) AND Y                          │
19560      X=CVI(MID$(T$[AP],J,2)) : J=J+2                   │
19570      WHILE X>&H232F AND X<&H233A AND Y                 │
19580       L=L*10+X-&H2330                                  │
19590       IF J>LEN(T$[AP]) THEN Y=FALSE                    │
19600       IF Y THEN X=CVI(MID$(T$[AP],J,2)): J=J+2         │
19610      WEND                                              │ 書式制御文字
19620      IF J>LEN(T$[AP]) THEN Y=FALSE                     │ の評価
19630      IF Y THEN X=CVI(MID$(T$[AP],J,2)) : J=J+2         │
19640      WHILE X>&H232F AND X<&H233A AND Y                 │
19650       R=R*10+X-&H2330                                  │
19660       IF J>=LEN(T$[AP]) THEN Y=FALSE                   ┘
```

```
19670        IF Y THEN X=CVI(MID$(T$[AP],J,2)) : J=J+2
19680       WEND
19690      WEND
19700      R=L+R : IF R=L THEN R=32767
19710      GOTO *LPT.1
19720  *LPT.0
19730      FOR J=3 TO LEN(T$[AP])-1 STEP 2                       ┐
19740       WHILE K<L : PRINT #2,MKI$(&H2021); : K=K+1 : WEND    │ファイルへの
19750       PRINT #2,MID$(T$[AP],J+1,1);MID$(T$[AP],J,1);        │書き出し
19760       K=K+1 : IF K>=R THEN K=0 : PRINT #2,                 ┘
19770      NEXT
19780      PRINT #2,
19790  *LPT.1
19800      GOSUB *NEXT.
19810      LC=LC+1
19820     WEND
19830     SL=I : CLOSE #2
19840     GOTO *MOVE.LINE
19850  '
19860  *READ.FILE
19870     IF SL=0 THEN SL=LC
19880     GOSUB *MOVE.LINE
19890     CC=CC+1 : A$=MID$(CM$,CC,1)
19900     IF A$="/" OR A$=" " THEN CC=CC+1 ELSE RETURN
19910     I=INSTR(CC,CM$,A$)
19920     IF I=0 THEN I=LEN(CM$)+1
19930     IF LEN(MID$(CM$,CC,I-CC))>0 THEN FI$=MID$(CM$,CC,I-CC)
19940     OPEN FI$ FOR INPUT AS #2
19950     IF LEN(CM$)>I THEN N = VAL( RIGHT$(CM$,LEN(CM$)-I)) ELSE N=0
19960     IF N=0 THEN N=32767
19970     WHILE N>0 AND NOT EOF(2)
19980      A$=INPUT$(2,2) : GOSUB *INSERT
19990      LAST=LAST+1 : LC=LC+1 : N=N-1
20000      WHILE CVI(A$)<>&HA0D
20010       T$[AP]=T$[AP]+MID$(A$,2,1)+MID$(A$,1,1)
20020       A$=INPUT$(2,2)
20030       IF LEN(T$[AP])<MAXLEN THEN *READ.0
20040        GOSUB *NEXT.
20050        GOSUB *INSERT
20060  *READ.0
20070      WEND
20080      GOSUB *NEXT.
20090     WEND
20100     CLOSE #2
20110     GOTO *MOVE.LINE
20120  '
20130  *WRITE.FILE
20140     IF SL=0 THEN SL=1
20150     IF EL=0 THEN EL=LAST
20160     CC=CC+2 : A$=MID$(CM$,CC-1,1)
20170     IF A$<>"/" AND A$<>" " THEN RETURN
20180     I=INSTR(CC,CM$,A$)
20190     IF I=0 THEN I=LEN(CM$)+1
20200     IF I>CC THEN FI$=MID$(CM$,CC,I-CC)
20210     CC=I+1 : A$=MID$(CM$,CC,1)
20220     IF A$="A" OR A$="a" THEN APPEND=TRUE ELSE APPEND=FALSE
20230     IF NOT APPEND THEN OPEN FI$ FOR OUTPUT AS #2
20240     IF APPEND THEN OPEN FI$ FOR APPEND AS #2
20250     L=0 : R=32767 : WF=TRUE : GOTO *PRINT.FILE
20260  '
20270  *VIDEO
20280     IF SL=0 THEN SL=LC
20290     CX=0 : CY=0 : DISP=TRUE
20300     GOSUB *MOVE.LINE
20310     RETURN *HOT
20320  '
20330  *DELETE.LINE
20340     IF SL=0 THEN SL=LC
20350     IF EL=0 THEN EL=SL
20360     GOSUB *CLR.Q.BUFF
20370     IF SL>EL THEN SWAP SL,EL
20380     IF SL=LAST THEN *MOVE.LINE
20390     IF EL<LAST THEN EL=EL+1
```

```
20400    GOSUB *MOVE.LINE
20410    WHILE SL<EL
20420     WHILE LEN(T$[AP])=MAXLEN
20430      GOSUB *SEVER
20440      MID$(T$[P],1,2)=MKI$(QP) : QP=P
20450     WEND
20460     SL=SL+1 : LAST=LAST-1 : GOSUB *SEVER
20470     MID$(T$[P],1,2)=MKI$(QP) : QP=P
20480    WEND
20490    RETURN
20500  '
20510  *YANK
20520    IF SL=0 THEN SL=LC
20530    IF EL=0 THEN EL=LC
20540    I=SL : IF SL>EL THEN SWAP SL,EL
20550    IF EL<LAST THEN EL=EL+1
20560    GOSUB *MOVE.LINE : GOSUB *CLR.Q.BUFF
20570    WHILE LC<EL
20580     GOSUB *YANK.Q.BUFF
20590     LC=LC+1 : GOSUB *NEXT.
20600    WEND
20610    SL=I : GOTO *MOVE.LINE
20620  '
20630  *PUT.
20640    IF SL=0 THEN SL=LC
20650    GOSUB *MOVE.LINE
20660    I=QP
20670    WHILE I<>EOT
20680      GOSUB *INSERT
20690      T$[AP]=T$[AP]+RIGHT$(T$[I],LEN(T$[I])-2)
20700      IF LEN(T$[AP])<MAXLEN THEN LAST=LAST+1
20710      I=CVI(T$[I])
20720    WEND
20730    RETURN
20740  '
20750  *TRAP
20760    IF ERR=68 THEN CLOSE #2 : PRINT 'Disk full'; : RESUME *TRAP1
20770    IF ERR=56 THEN *CATALOG
20780    IF ERR=69 THEN PRINT 'Bad allocation table'; : RESUME *TRAP1
20790    IF ERR=64 THEN PRINT CLOSE #2:PRINT 'Disk I/O error';:RESUME *
TRAP1
20800    IF ERR=53 THEN *CATALOG
20810    IF ERR=61 THEN PRINT 'file write protected'; : GOTO *TRAP1
20820    IF ERR=9 OR ERR=14 THEN RESUME *TRAP0
20830    ON ERROR GOTO 0
20840  *TRAP0
20850    PRINT 'text is too large' : GOSUB *CLR.Q.BUFF : RETURN *HOT
20860  '
20870  *CATALOG
20880    SCREEN ,3
20890    CONSOLE 0,20 : CLS
20900    N=VAL(FI$) : IF N=0 THEN N=1
20910    FILES N : PRINT : PRINT ' hit any key'; : V$=INPUT$(1,1)
20920    CLS : CONSOLE CMNDLINE,20-CMNDLINE : CLS
20930    SCREEN ,0 : RESUME *TRAP1
20940  *TRAP1 RETURN *LINE.EDIT
20950  '          漢字用データ(読み, 漢字コード, 個数)
20960  *KYWORD DATA ,8481,108,ｱﾜ,12352,1,ｳﾏﾔ,12601,1,ｵﾝ,12856,4
20970    DATA ｶﾔ,13180,3,ｷﾞ,13622,15,ｹﾂ,14182,8,ｻｲ,14916,26
20980    DATA ｼｬｸ,15450,9,ｼﾞｬ,15448,2,ｽﾝ,16419,1,ﾀｲ,16974,23
20990    DATA ﾀﾞﾝ,17220,9,ﾂﾌﾞ,17497,1,ﾄﾘ,18003,1,ﾆｮ,18209,1
21000    DATA ﾊｳ,18279,1,ﾊﾞﾝ,18516,7,ﾌｳ,18805,3,ﾎﾘ,19289,1,ﾏﾝ,19324,5
21010    DATA ﾓﾄﾞ,19553,1,ﾘｬｸ,20011,2,ﾜﾆ,20300,1,ﾜﾝ,20304,1
21020  *K0 RESTORE *K0 : RETURN
21030    DATA G,1280,24,R,256,33,g,-224,24,r,271,33,ｱ,2257,4,ｱｲ,4,3
21040    DATA ｱｳ,3,2,ｱｵｲ,2,1,ｱｶﾈ,1,1,ｱｷ,1,1,ｱｸ,1,3,ｱｻﾋ,3,1,ｱｼ,1,2
21050    DATA ｱｼﾞ,2,1,ｱｽﾞｻ,1,1,ｱﾂ,1,3,ｱﾃ,3,1,ｱﾈ,1,1,ｱﾌﾞ,1,1,ｱﾒ,1,1
21060    DATA ｱﾔ,1,2,ｱﾕ,2,1,ｱﾙ,1,1
21070  *K1 RESTORE *K1 : RETURN
21080    DATA ｱﾜｾ,1,1,ｱﾝ,1,8,ｲ,8,30,ｲｷ,30,1,ｲｸ,1,2,ｲｿ,2,1,ｲﾁ,1,2
21090    DATA ｲﾂ,2,2,ｲﾈ,2,1,ｲﾊﾞﾗ,1,1,ｲﾓ,1,1,ｲﾜｼ,1,1,ｲﾝ,1,16,ｳ,178,8
21100    DATA ｳｶｶﾞｳ,8,1,ｳｼ,1,1,ｳｽ,1,2,ｳｽﾞ,2,1,ｳｿ,1,1,ｳﾀ,1,1,ｳﾂ,1,2
21110    DATA ｳﾅｷﾞ,2,1,ｳﾊﾞ,1,1
```

```
21120 *K2 RESTORE *K2 : RETURN
21130   DATA ｳﾗ,1,1,ｳﾘ,1,1,ｳﾙｳ,1,1,ｳﾜｻ,1,1,ｳﾝ,1,3,ｴ,3,2,ｴｲ,2,18
21140   DATA ｴｷ,18,4,ｴﾂ,4,4,ｴﾉｷ,4,1,ｴﾝ,1,25,ｵ,25,2,ｵｲ,2,1,ｵｳ,1,16
21150   DATA ｵｶ,178,1,ｵｷ,1,1,ｵｷﾞ,1,1,ｵｸ,1,4,ｵｹ,4,1,ｵｽ,1,1,ｵﾂ,1,1
21160   DATA ｵﾚ,1,1,ｵﾛｼ,1,1
21170 *K3 RESTORE *K3 : RETURN
21180   DATA ｶ,4,40,ｶｲ,52,28,ｶｲﾘ,205,1,ｶｴﾙ,2,1,ｶｵﾘ,-1,1,ｶｷ,2,3
21190   DATA ｶｷﾞ,3,1,ｶｸ,1,20,ｶｹ,25,1,ｶｻ,1,1,ｶｼ,1,2,ｶｼﾞ,2,1,ｶｼﾞｶ,1,1
21200   DATA ｶﾂ,2,11,ｶﾂｵ,11,1,ｶﾅｳ,1,1,ｶﾊﾞ,2,1,ｶﾊﾞﾝ,1,1,ｶﾌﾞ,1,1
21210   DATA ｶﾌﾞﾄ,1,1,ｶﾏ,1,4,ｶﾑ,4,1,ｶﾓ,1,1
21220 *K4 RESTORE *K4 : RETURN
21230   DATA ｶｺ,165,1,ｶﾘ,1,2,ｶﾜﾗ,3,1,ｶﾝ,1,56,ｶﾞ,-450,12,ｶﾞｲ,202,15
21240   DATA ｶﾞｸ,42,5,ｶﾞﾀ,11,1,ｶﾞﾝ,250,14,ｷ,14,41,ｷｬｸ,230,3,ｷｭｳ,5,23
21250   DATA ｷｮ,24,11,ｷｮｳ,14,38,ｷｮｸ,205,3,ｷｸ,-260,3,ｷﾂ,3,6,ｷﾇ,6,1
21260   DATA ｷﾈ,1,1,ｷﾋﾞ,1,1,ｷﾘ,253,1,ｷﾛ,1,1,ｷﾝ,1,20
21270 *K5 RESTORE *K5 : RETURN
21280   DATA ｷﾞｭｳ,55,1,ｷﾞｮ,12,3,ｷﾞｮｳ,203,5,ｷﾞｮｸ,8,1,ｷﾞｬｸ,-248,2
21290   DATA ｷﾞﾝ,271,2,ｸ,2,12,ｸｳ,15,2,ｸｼ,6,3,ｸｽﾞ,3,1,ｸﾂ,1,6
21300   DATA ｸﾎﾞ,168,1,ｸﾏ,1,2,ｸﾒ,2,1,ｸﾘ,1,2,ｸﾜ,2,2,ｸﾝ,18,4,ｸﾞ,-205,3
21310   DATA ｸﾞｳ,5,4,ｸﾞﾝ,188,3,ｹ,3,3,ｹｲ,3,37,ｹﾀ,45,1
21320 *K6 RESTORE *K6 : RETURN
21330   DATA ｹﾝ,9,36,ｹﾞｲ,-18,3,ｹﾞｷ,3,5,ｹﾞﾂ,14,1,ｹﾞﾝ,199,14,ｺ,14,27
21340   DATA ｺｲ,46,1,ｺｳ,1,80,ｺｳｼﾞ,251,1,ｺｸ,1,8,ｺｼ,10,1,ｺｼｷ,1,11
21350   DATA ｺｽ,-2,1,ｺﾂ,3,3,ｺﾏ,3,1,ｺﾑ,1,1,ｺﾚ,163,1,ｺﾛ,1,1,ｺﾝ,1,16
21360   DATA ｺﾞ,-453,19,ｺﾞｳ,263,8,ｺﾞｸ,17,1,ｻ,189,14
21370 *K7 RESTORE *K7 : RETURN
21380   DATA ｻｴ,31,1,ｻｶ,1,3,ｻｶｷ,3,1,ｻｶﾅ,1,1,ｻｷ,1,4,ｻｷﾞ,4,1,ｻｸ,1,11
21390   DATA ｻｸﾗ,11,1,ｻｹ,1,1,ｻｻ,1,1,ｻｼﾞ,1,1,ｻﾂ,1,9,ｻﾊﾞ,173,2,ｻﾋﾞ,2,1
21400   DATA ｻﾒ,1,1,ｻﾗ,1,2,ｻﾝ,2,18,ｻﾞ,-239,3,ｻﾞｲ,29,5,ｻﾞﾂ,202,2
21410   DATA ｻﾞﾝ,26,3,ｼ,3,49,ｼｬ,262,12
21420 *K8 RESTORE *K8 : RETURN
21430   DATA ｼｭ,13,13,ｼｭｳ,21,28,ｼｭｸ,203,7,ｼｭﾂ,9,1,ｼｭﾝ,3,7,ｼｮ,21,13
21440   DATA ｼｮｳ,21,70,ｼｮｸ,255,13,ｼｵ,-590,1,ｼｶ,1,1,ｼｷ,1,2,ｼｷﾞ,2,1
21450   DATA ｼｽﾞｸ,4,1,ｼﾁ,1,1,ｼﾂ,1,10,ｼﾄﾐ,11,1,ｼﾉ,1,2,ｼﾊﾞ,2,3,ｼﾍﾞ,3,1
21460   DATA ｼﾏ,1,1,ｼﾘ,738,1,ｼﾝ,1,32,ｼﾞ,-951,22
21470 *K9 RESTORE *K9 : RETURN
21480   DATA ｼﾞｬｸ,11,4,ｼﾞｭ,17,8,ｼﾞｭｳ,198,13,ｼﾞｭｸ,20,2,ｼﾞｭﾂ,3,2
21490   DATA ｼﾞｭﾝ,9,14,ｼﾞｮ,27,8,ｼﾞｮｳ,240,23,ｼﾞｮｸ,198,1,ｼﾞｸ,-760,3
21500   DATA ｼﾞﾂ,15,1,ｼﾞﾝ,779,13,ｽ,13,4,ｽｲ,7,14,ｽｳ,17,6,ｽｴ,6,1
21510   DATA ｽｷﾞ,1,2,ｽｹﾞ,2,1,ｽｺ,1,1,ｽｽﾞﾒ,1,1,ｽｿ,1,1,ｽﾑ,163,1,ｽﾙ,1,1
21520 *K10 RESTORE *K10 : RETURN
21530   DATA ｽﾞ,-197,3,ｽﾞｲ,17,3,ｾ,181,3,ｾｲ,4,31,ｾｷ,33,18,ｾﾂ,17,10
21540   DATA ｾﾐ,12,1,ｾﾝ,1,39,ｾﾞ,-64,1,ｾﾞｲ,32,2,ｾﾞﾂ,29,2,ｾﾞﾝ,204,9
21550   DATA ｿ,9,21,ｿｳ,21,49,ｿｸ,215,10,ｿﾂ,15,1,ｿﾃﾞ,1,1,ｿﾉ,1,1,ｿﾛ,1,1
21560   DATA ｿﾝ,1,6,ｿﾞｳ,-23,4,ｿﾞｸ,14,5,ﾀ,15,5
21570 *K11 RESTORE *K11 : RETURN
21580   DATA ﾀｶ,29,1,ﾀｷ,1,2,ﾀｸ,2,10,ﾀｹ,13,1,ﾀｺ,1,2,ﾀﾀｸ,165,1
21590   DATA ﾀﾀﾞ,-163,1,ﾀﾀﾞｼ,164,1,ﾀﾂ,1,2,ﾀﾂﾐ,4,1,ﾀﾃ,1,1,ﾀﾄﾞ,1,1
21600   DATA ﾀﾅ,1,1,ﾀﾆ,1,1,ﾀﾇｷ,1,1,ﾀﾗ,1,1,ﾀﾙ,1,1,ﾀﾝ,2,20,ﾀﾞ,-237,11
21610   DATA ﾀﾞｲ,34,6,ﾀﾞｸ,20,2,ﾀﾞﾂ,172,2,ﾀﾞﾚ,10,1
21620 *K12 RESTORE *K12 : RETURN
21630   DATA ﾁ,9,14,ﾁｬ,22,3,ﾁｭｳ,3,16,ﾁｮ,16,4,ﾁｮｳ,4,34,ﾁｮｸ,194,3
21640   DATA ﾁｸ,-225,6,ﾁﾂ,6,2,ﾁﾝ,-18,6,ﾂ,246,1,ﾂｲ,1,5,ﾂｳ,5,2,ﾂｶ,2,1
21650   DATA ﾂｶﾑ,2,1,ﾂｶﾞ,-1,1,ﾂｷ,2,1,ﾂｸﾀﾞ,1,1,ﾂｹ,1,2,ﾂｼﾞ,2,1,ﾂﾀ,1,1
21660   DATA ﾂﾂﾞ,1,1,ﾂﾊﾞ,1,1,ﾂﾊﾞｷ,1,1
21670 *K13 RESTORE *K13 : RETURN
21680   DATA ﾂﾎﾞ,1,1,ﾂﾏ,2,1,ﾂﾑ,1,1,ﾂﾒ,1,1,ﾂﾙ,1,3,ﾃｲ,3,33,ﾃｷ,196,8
21690   DATA ﾃﾂ,9,6,ﾃﾝ,6,12,ﾃﾞｲ,-16,1,ﾃﾞｷ,9,1,ﾃﾞﾝ,19,5,ﾄ,5,18
21700   DATA ﾄｳ,23,48,ﾄｳｹﾞ,224,1,ﾄｷ,1,1,ﾄｸ,1,8,ﾄﾁ,11,2,ﾄﾂ,1,3
21710   DATA ﾄﾄﾞ,4,1,ﾄﾋﾞ,1,1,ﾄﾏ,1,1,ﾄﾗ,1,1
21720 *K14 RESTORE *K14 : RETURN
21730   DATA ﾄﾛ,1,1,ﾄﾝ,1,8,ﾄﾞ,-253,5,ﾄﾞｳ,215,14,ﾄﾞｸ,24,3,ﾄﾞﾝ,22,3
21740   DATA ﾅ,3,2,ﾅｲ,2,1,ﾅｶﾞ,1,1,ﾅｷﾞ,1,1,ﾅｸﾞ,1,1,ﾅｿﾞ,1,1,ﾅﾀﾞ,1,1
21750   DATA ﾅﾂ,1,1,ﾅﾍﾞ,1,1,ﾅﾗ,1,1,ﾅﾗｽ,1,1,ﾅﾜ,1,1,ﾅﾜﾃ,1,1,ﾅﾝ,1,4
21760   DATA ﾅﾝｼﾞ,4,1,ﾆ,1,4,ﾆｭｳ,10,2
21770 *K15 RESTORE *K15 : RETURN
21780   DATA ﾆｮｳ,1,1,ﾆｵ,-171,1,ﾆｷﾞ,1,1,ﾆｸ,1,1,ﾆｼﾞ,1,1,ﾆｼﾞｭｳ,1,1
21790   DATA ﾆﾁ,1,1,ﾆﾗ,167,1,ﾆﾝ,1,4,ﾇ,4,1,ﾈ,1,2,ﾈｲ,2,1,ﾈｷﾞ,1,1
21800   DATA ﾈｺ,1,1,ﾈﾂ,1,1,ﾈﾝ,1,6,ﾉ,6,4,ﾉｳ,4,8,ﾉｿﾞ,8,1,ﾉﾐ,1,1,ﾊ,1,9
21810   DATA ﾊｬﾌﾞｻ,248,1,ﾊｲ,-235,12
21820 *K16 RESTORE *K16 : RETURN
21830   DATA ﾊｴ,1,2,ﾊｷﾞ,2,1,ﾊｸ,2,12,ﾊｺ,-75,2,ﾊｻﾞﾏ,2,1,ﾊｼ,1,1
```

```
21840    DATA ﾊｼﾞﾒ,1,1,ﾊｽﾞ,1,1,ﾊｾﾞ,1,1,ﾊﾀ,1,1,ﾊﾀｹ,2,2,ﾊﾀﾞ,-1,1,ﾊﾁ,3,2
21850    DATA ﾊﾂ,2,4,ﾊﾄ,265,1,ﾊﾅｼ,1,1,ﾊﾅﾜ,1,1,ﾊﾏｸﾞﾘ,1,1,ﾊﾝ,2,24
21860    DATA ﾊﾞ,-240,4,ﾊﾞｲ,16,11,ﾊﾞｸ,28,7,ﾊﾞﾂ,186,5
21870  *K17 RESTORE *K17 : RETURN
21880    DATA ﾋ,7,28,ﾋｬｸ,217,1,ﾋｮｳ,2,10,ﾋｲﾗｷﾞ,-20,1,ﾋｴ,1,1,ﾋｷ,1,2
21890    DATA ﾋｹﾞ,2,1,ﾋｺ,1,1,ﾋｻﾞ,1,1,ﾋｼ,1,1,ﾋｼﾞ,1,1,ﾋﾂ,1,5,ﾋﾉｷ,5,1
21900    DATA ﾋﾒ,1,2,ﾋﾓ,2,1,ﾋﾙ,20,2,ﾋﾚ,2,1,ﾋﾝ,1,8,ﾋﾞ,-211,9
21910    DATA ﾋﾞｭｳ,190,1,ﾋﾞｮｳ,11,7,ﾋﾞﾝ,18,2,7,2,26
21920  *K18 RESTORE *K18 : RETURN
21930    DATA ﾌｷ,3,2,ﾌｸ,2,9,ﾌﾁ,171,1,ﾌﾂ,1,4,ﾌﾅ,5,1,ﾌﾝ,1,12,ﾌﾞ,-190,7
21940    DATA ﾌﾞﾂ,188,1,ﾌﾞﾝ,14,2,ﾍｲ,2,12,ﾍｷ,14,4,ﾍﾗ,7,1,ﾍﾝ,1,8
21950    DATA ﾍﾞｲ,-10,1,ﾍﾞﾂ,6,3,ﾍﾞﾝ,12,5,ﾍﾟｲｼﾞ,-17,1,ﾎ,22,10,ﾎｳ,18,34
21960    DATA ﾎｴ,219,1,ﾎｵ,1,1,ﾎｸ,1,1,ﾎﾄ,12,1
21970  *K19 RESTORE *K19 : RETURN
21980    DATA ﾎﾛ,1,1,ﾎﾝ,1,3,ﾎﾞ,-244,8,ﾎﾞｳ,204,23,ﾎﾞｸ,26,8,ﾎﾞﾀﾝ,8,1
21990    DATA ﾎﾞﾂ,1,2,ﾎﾞﾝ,8,2,ﾏ,2,4,ﾏｲ,4,5,ﾏｲﾙ,5,1,ﾏｷ,1,1,ﾏｸ,1,2
22000    DATA ﾏｸﾗ,2,1,ﾏｸﾞﾛ,1,1,ﾏｻ,1,1,ﾏｽ,1,2,ﾏﾀ,2,3,ﾏﾂ,3,3,ﾏﾃﾞ,3,1
22010    DATA ﾏﾏ,1,1,ﾏｺ,1,1,ﾏﾛ,1,1
22020  *K20 RESTORE *K20 : RETURN
22030    DATA ﾐ,167,5,ﾐｬｸ,11,1,ﾐｮｳ,1,1,ﾐｻｷ,-7,1,ﾐﾂ,1,2,ﾐﾅﾄ,2,1,ﾐﾉ,1,1
22040    DATA ﾐﾉﾙ,1,1,ﾐﾘ,3,1,ﾐﾝ,1,2,ﾑ,2,7,ﾑｸ,7,1,ﾑｺ,1,1,ﾑｽﾒ,1,1
22050    DATA ﾒｲ,1,9,ﾒｽ,9,1,ﾒﾂ,1,1,ﾒﾝ,1,6,ﾓ,6,3,ﾓｯﾄﾓ,18,1,ﾓｳ,-15,9
22060    DATA ﾓｸ,9,4,ﾓﾁ,4,2
22070  *K21 RESTORE *K21 : RETURN
22080    DATA ﾓﾐ,1,1,ﾓﾐｼﾞ,-6385,1,ﾓﾗ,6386,1,ﾓﾝ,1,4,ﾓﾝﾒ,4,1,ﾔ,1,8
22090    DATA ﾔｸ,8,6,ﾔｽ,6,1,ﾔﾅｷﾞ,1,1,ﾔﾌﾞ,1,1,ﾔﾘ,1,1,ﾕ,1,6,ﾕｲ,168,1
22100    DATA ﾕｳ,1,25,ﾖ,25,6,ﾖｳ,6,26,ﾖｸ,26,7,ﾖﾄﾞ,7,1,ﾗ,1,3,ﾗｲ,3,4
22110    DATA ﾗｸ,4,4,ﾗﾝ,4,8,ﾘ,8,12
22120  *K22 RESTORE *K22 : RETURN
22130    DATA ﾘｭｳ,2,9,ﾘｮ,10,4,ﾘｮｳ,10,20,ﾘｮｸ,14,2,ﾘｸ,-41,1,ﾘﾂ,1,4
22140    DATA ﾘﾝ,42,11,ﾙ,11,1,ﾙｲ,1,4,ﾚｲ,4,16,ﾚｷ,16,2,ﾚﾂ,2,4,ﾚﾝ,4,11
22150    DATA ﾛ,173,7,ﾛｳ,7,16,ﾛｸ,16,5,ﾛﾝ,5,1,ﾜ,1,3,ﾜｲ,3,2,ﾜｷ,2,1
22160    DATA ﾜｸ,1,2,ﾜｼ,2,1,ﾜﾀ,1,2
22170  *K23 RESTORE *K23 : RETURN
22180    DATA ﾜﾋﾞ,1,1,ﾜﾗ,1,1,ﾜﾗﾋﾞ,1,1,ﾜﾝ,1,1,ﾝ,,,ﾝ,,,ﾝ,,,ﾝ,,
22190    DATA ﾝ,,,ﾝ,,,ﾝ,,,ﾝ,,,ﾝ,,,ﾝ,,,ﾝ,,,ﾝ,,
22200    DATA ﾝ,,,ﾝ,,,ﾝ,,,ﾝ,,,ﾝ,,,ﾝ,,,ﾝ,,,ﾝ,,
```

0. 40
スクリーン・エディターのコマンド一覧表
CTRL-A HELP　カーソルのある行に関する直前の変更を無効にする。
CTRL-B f・6　直前の語の語尾にカーソルを移動する。
CTRL-C STOP　入力可能な行数と文字数を表示する。
CTRL-D　ライン・エディター・モードに入る。
CTRL-E　カーソルのある位置から行末までを消去する。
CTRL-F f・7　次の語の先頭にカーソルを移動する。

ESC+ヱ　旧カナづかいの「ヰ」および「ゐ」，または「ヱ」および「ゑ」を入力します。
ESC+数字+コントロール・キャラクター　数字の回数だけ，そのコマンドをくりかえします。
ESC+数字+ESC+文字列+ESC　数字の回数だけ，ESC間に伏まれた文字列をくりかえし入力します。文字はコントロール・キャラクターでもかまいません。
ESC+数字+英文字列+スペース　登録は100個まで可能です。
ESC+英文字列+スペース　英文字列の名前で登録済の文字を入力します。
ESC+英文字列+DEL　英文字列の名前での登

ESC シーケンス の説明
^10, 40
スクリーン・エディターのコマンド一覧表
CTRL-A HELP　カーソルのある行に関する直前の変更を無効にする。
CTRL-B　直前の語の語尾にカーソルを移動する。
1,$

DEL　カーソルの下の文字を削除する。ただし行末の場合は，次の行と合わせて1つ行にする。
→　カーソルを次の文字へ移動させる。
←　カーソルを前の文字へ移動させる。
↑　カーソルを上のラインへ移動させる。
↓　カーソルを下のラインへ移動させる。
ESC シーケンス の説明

5.3.2 グラフへの漢字の導入

5．1で説明したグラフの各サブルーチンにおいて，漢字タイトルは，メインルーチン中のDATA文によって設定されています．そして，その値を漢字コードとし整数配列 TITLE% を介して，各サブルーチンに渡されて漢字が表示されていました．

ここでは，この漢字を表示するサブルーチン＊TITLE について説明しましょう．これは，各ライブラリ・サブルーチンの内部で呼ばれており，ユーザーが直接呼ぶ必要はありません．このサブルーチンを実行すると，漢字タイトルが最上位行の3文字目から表示されます．タイトルの長さは最大24文字で，24文字表示し終るか，&H0D0A のコードが出てくることにより RETURN します．これ以外のときは，漢字の PUT＠文を実行し，漢字を表示します．PUT＠文の座標はスクリーン座標ですから，VIEW 文の実行後に行うと，場所がずれてしまいますから，このサブルーチンは VIEW 文の実行前に呼ぶ必要があります．スクリーン座標の値は，Yが一定で0のまま，Xが1文字表示した後，右に16ドットずれていくだけです．Xに32を加えているのは3文字目から表示するためです．

次に，このサブルーチンに漢字の並びを引き渡す配列，TITLE% について説明しましょう．漢字コードは図9のようなフォーマットで渡されます．要するに，&H0D0A というコードまで，順番に並んでいるのです．また，最大の24文字を表示するときでも，添字は23までですが，終了判定の IF 文で，添字の次の値のときの配列を参照するため，配列要素は1つ余分に取ってあります．このとき，添字が24になる要素に &H0D0A を代入しておく必要はありません．そして，当然ですが，添字が0の要素に &H0D0A を代入しておけば，何も表示せずに終了します．もし漢字 ROM を持っていない方はこの処置をしてください．

TITLE(0)	1文字目の漢字コード	
TITLE(1)	2文字目の漢字コード	
TITLE(2)	3文字目の漢字コード	
	・ ・ ・	
TITLE(n)	n+1文字目の漢字コード	
TITLE(n+1)	& H0D0A	←終了コード
	・ ・ ・	
TITLE(24)		

図9　漢字ファイルのフォーマット

それでは，このサブルーチンに漢字ワード・プロセッサで作ったデータを渡す方法について説明しましょう．

漢字ワード・プロセッサは，ディスク上にファイルを作成できますから，これを読み出して利用します．従って，ディスクをお持ちでない方は残念ながら実行できません．ディスクからタイ

トルを読み出して表示するサブルーチン＊TITLE．DSKをリスト8に示します．＊TITLEをこれに差し換えることにより，タイトル表示はディスク上のファイルから行う様になります．

リスト8

```
65000 *TITLE.DSK
65010 '
65020 '  PRINT TITLE
65030 '     from DISK FILE
65040 OPEN TITLE.NAM$ FOR INPUT AS #1
65050 PT=0
65060 WHILE NOT EOF(1) AND (TITLE%(PT)<>&HD0A) AND PT<24
65070   TITLE%(PT)=CVI(INPUT$(2,#1))
65080   PT=PT+1
65090 WEND
65100 PT=0
65110 WHILE (TITLE%(PT)<>&HD0A) AND (PT<24)
65120   PUT (32+PT*16,0),KANJI(TITLE%(PT))
65130   PT=PT+1
65140 WEND
65150 RETURN
```

漢字コードは，ディスク上のファイルでは2バイトに分けられ，それぞれが文字コードとして収められています．つまり，2文字で漢字1文字のコードとなる訳で，1文字目が上位バイト，2文字目が下位バイトの値です．また，このファイルは復帰，改行コード（&H0D，&H0A）で終了しているので，終了条件も前のサブルーチンと同じです．もし，ファイルに漢字25文字以上が入っていた場合は，先頭の24文字のみを表示し残りは無視します．

このサブルーチンを使うには，タイトルの漢字コードの収まっているファイル名を指定しなければなりません．ファイル名は，TITLE．NAM$という文字変数に代入して引き渡します．ファイル・バッファは#1を使っていますが，もし，これがすでに使われている場合は，別のバッファに変更してください．

6章　静的グラフィックス

本章では，２，３章で解説したことや４章で紹介したツールを使って，色々な応用プログラムを組んでみます．動的なものは次章で扱うことにして，静的なものを中心にお話しします．

静的グラフィックスと言っても色々なものがありますが，主なものとしては，デザインや絵などが挙げられます．このデザインや絵は，基本的には２章で解説した，形や色の操作の応用です．私達が紙面上でこのような操作を行うことは，割と簡単にできますが，画面上で操作を行うにはそれなりの技術が必要です．本章では，これらのことをプログラムを実際に示しながら解説していきます．

また，このようなコンピュータ・アートは，現在大変盛んになってきました．専用機と比べると不利な点を多く持っているローコスト・コンピュータでは，専用機のようにはいきませんが，プログラムでうまくカバーすれば，それに近いものは実現できるものです．特に，N_{88}-BASICのような多彩な機能を持ったものは，かなり近いところまでいくのではないでしょうか．皆さんもこの章を参考にして，自分なりのソフトを考えてみてください．

6.1　白黒3ページ・モードを使った部屋のレイアウト(Room)

まず初めに，部屋のレイアウトを題材に取って話を進めていきましょう．私たちが部屋に物を置く時には，一般に，大まかな部屋の大きさや物の大きさを考え，紙などを用いながら位置を定めていきます．しかし，紙を用いる場合には，線が曲がってしまったり，書いたものを消す時に消したくない所までも消してしまったり，また，紙を汚してしまったりで，なかなかうまくいかないものです．そこで，今回は紙の代わりに画面を用いて，部屋のレイアウトに挑戦してみます．では，どのような方法で実現していくのかを，例を挙げながら順を追って説明しましょう．

6.1.1　画面上の位置指定

まず一番問題になるのが，画面上の任意の位置をいかにして指定するかということです．簡単な方法としては，ライト・ペンやデジタイザを用いる方法があります．しかし，この方法は，コ

ストの面などを考えると一般的ではありません．それで，このような機能を持つものをソフトでカバーしていかなければなりません．そこで，考えられるのが，第1巻のテレビ黒板で紹介されている**十字カーソル**です．これは，キーボードにより操作が可能で，ドット単位の指定ができるため大変便利です．

ではここで，十字カーソルの作り方について説明しましょう．まず，リスト1の*CURSORINITを見てください．この部分が，十字カーソルを作るルーチンであり，まず十字カーソルをLINE文で描き，これをGET@命令で拾っています．そして，ここで作った十字カーソルのパターンをPUT@で順次置いていき，十字カーソルの移動を行います．この十字カーソルの移動の処理を行っているのが*MOVECURSORです．移動の方法はPUT@の〝XOR〞の機能を利用して，現在の十字カーソルの上に，もう一度XORで置き，そのカーソルを消去してから，次の場所に十字カーソルのパターンを置くという処理をしています．これで，見かけ上十字カーソルが移動しているように見える訳です（図1）．さらに，XORでPUTしているため，画面上で既にパターンが存在していたとしても，十字カーソルの移動によりそのパターンが壊されることもありません（図2）．

また，十字カーソルの操作の仕方は，カーソル・キーを用いており，方向はカーソル・キーの方向に準拠するようにしてあります．十字カーソルの移動幅は，いくつか用意しておくと便利です．今回の移動幅は，*CDATAの通りです．

では次に，この十字カーソルを使って，今回の目的である画面上にいかにして物を設置していくかということについて説明しましょう．部屋に置くものは，一般に四角の物が多いため，今回は四角に限定してしまいます．まず四角の物を画面上に置くには2つの方法が考えられます．1

図1　十字カーソルの作り方と動かし方

図2　十字カーソルを置いても元の図に影響がない理由

図3　2点指定による決定

図4　1点指定による決定

つには，2点を指定する方法です（図3）．この2点指定は，四角の大きさが決定してしまうと，四角を描くベクトルが可変であるため，位置指定も簡単に行えます．またプログラムにおいてもLINE（X_0，Y_0）－（X，Y），Bと単純です．もう1つには，1点指定があります（図4）．この方法は，大きさを始めに定義し四角の形を決定しておく方法で，位置指定は四角を描くベクトルが不変であるため，場合によっては，物を置く目的の位置まで移動しなければなりません．物の移動は，一般的には十字カーソルの様にPUT＠でXORの機能を用いて描いたり，消したりして行います．しかし，この方法では，位置指定が多くの場合，1回では決まらないために複雑になってしまいます．以上の理由により今回は，前者の方法を採りました．また，今回のように2点を指定する時には，2点間の距離の目安として，x軸（水平）方向とy軸（垂直）方向にそれぞれ目盛を用意しておくと，十字カーソルのSTEP数（移動量）も（cm）単位にしておくと大変便利で，入力が仕易くなります．

では次に，置いた物を消す時に注意しなければならない点について考えてみましょう．

6.1.2 画面の合成（3画面の使用）

画面上に物を描いて，それを消す時に生じる問題が，重なっている部分も一緒に消えてしまうという点です．今回の場合，タタミの目の上に物を置きそれを消してしまうと，タタミの目まで一緒に消えてしまうことになります．簡単なパターンならば，消えた所を置き直せばよいのですが，大きなものや複雑なパターンでは時間がかかってしまいます．そこで今回は，画面の合成というテクニックを用います．図5を見て頂ければ分かると思いますが，第1画面にタタミを描き，第3画面に目盛と部屋の枠を描きます．そして，物を置いたり消したりする時には，第2画面をアクティブにしておけば，タタミの目などは消えることなく，物だけを消すことができる訳です．また，画面の表示の仕方も色々変えることができます．今回は，物を置く時には，（2画面＋3画面），コマンド待ちの時には，（1画面＋2画面）としました．

では次に，実際のプログラム（リスト1）について基本の部分を説明していきましょう．

6.1.3 プログラムの説明

今回，部屋は4帖半と6帖を用意しました．タタミは，共にLINE命令で描いています．注意する点は，白黒3ページ・モードですから，縦・横比（2：1）であることです．また，物を置いた時に区別を付けるために，16個の（8×4）ドットのタイリング・パターンを用意しました．

〔1画面〕 〔2画面〕 〔3画面〕

〔合成〕

図5 マルチ・ページの使用とその合成

8 (HEX)

4

11
22
44
88

図6 タイル・ストリングの作り方

このパターンは，図6の様にして拾いました（4章の〝タイル・パターン・ジェネレータ〟を用いてもよいのですが，この程度のパターンですと手で拾った方が早い）.

＊MOVEにおけるIF文は，物を置く時に，物と物とが重ならないように，十字カーソルが物から1ドットずれるようになっています．また，漢字ROMを使用していない方は，部屋の大きさの表示が出力されませんので注意してください．他の説明については，リスト1の右側のコメントを参考にしてください．

リスト1

```
1000 '
1010 '   room  (sample)  [ (640 x 200) x 3 mode ]
1020 '        ver 3.1        By A.Tamura
1030 '
1040 '
1050 '
1060 ON ERROR GOTO *ERRTRAP
1070 '
1080 ON STOP GOSUB *ABORT
1090 STOP ON
1100 '
1110 SCREEN 0,3:WIDTH 80,25:CONSOLE 0,25,0,1
1120 CLS 3:SCREEN 1,0,0,1:DIM CU%(12),INCX(6),INCY(6)
1130 X=200:Y=100:C=0:N=0:MC=0
1140 GOSUB *CURSORINIT
1150 '
1160 *SELECT
```

↑ Gカーソル（CU%(12)）

Gカーソルの STEP 数（X軸，Y軸方向）（INCX(6), INCY(6)）

```
1170   CLS:INPUT "4.5 ｼﾞｮｳ (1) or 6 ｼﾞｮｳ (2)";A
1180   IF (A<>1) AND (A<>2) THEN *SELECT
1190   ON A GOSUB *TATAMI1,*TATAMI2
1200   POINT (10,10)
1210   SCREEN 1,0,1,7 ←――――3画面を合成して表示
1220   ON S GOSUB *DISP1,*DISP2
1230   GOSUB *BOXINIT
1240  '
1250  *LOOP1
1260   CONSOLE 0,22:CLS:MC=0
1270   COLOR@(5,23)-(65,24),7
1280   SCREEN ,,,3 ←――――第1，2画面を表示
1290   GOSUB *SELECT1
1300   A$=INPUT$(1)
1310   IF A$="m" THEN GOSUB *MOVE
1320   IF A$="d" THEN GOSUB *DELETE1
1330   IF A$="c" THEN GOSUB *SCLS
1340   IF A$="h" THEN GOSUB *GCOPY
1350  GOTO *LOOP1
1360  '
1370  *MOVE (mコマンド)
1380   CLS:COLOR@(5,23)-(20,23),3
1390   ON S GOSUB *DISP1,*DISP2
1400   SCREEN 1,0,1,6 ←―― 第2，3画面を表示
1410   GOSUB *MOVECUR
1420   PSET (X,Y)
1430   X0=X:Y0=Y
1440   GOSUB *MOVECUR
1450   GOSUB *BOX
1460   IF X=100 AND Y=30 THEN X=X0+1:Y=Y0+1
1470   IF X=100 AND Y=162 THEN X=X0+1:Y=Y0-1
1480   IF (X=364 OR X=452) AND Y=30 THEN X=X0-1:Y=Y0+1
1490   IF (X=364 OR X=452) AND Y=162 THEN X=X0-1:Y=Y0-1
1500   IF X>X0 AND Y>Y0 THEN X=X+1:Y=Y+1
1510   IF X<X0 AND Y<Y0 THEN X=X-1:Y=Y-1
1520   IF X>X0 AND Y<Y0 THEN X=X+1:Y=Y-1
1530   IF X<X0 AND Y>Y0 THEN X=X-1:Y=Y+1
1540   GOSUB *LIMIT
1550  RETURN
1560  '
1570  *BOX
1580   IF X0=X OR Y0=Y THEN PRESET (X0.Y0):RETURN
1590   LINE (X0,Y0)-(X,Y),7,B ←― 物を置く(四角を描く)
1600   GOSUB *TILEING
1610   N=N+1 ←――――――― 置いた物の数をカウントする
1620  RETURN
1630  '
1640  *TATAMI1
1650   POINT (100,30):CLS
1660   LINE-STEP(88,88),7,B ←― 4.5帖のたたみを描く
1670   LINE STEP(-88,0)-STEP(176,44),7,B
1680   LINE-STEP(88,-88),7,B
1690   LINE-STEP(-176,-44),7,B
1700   GOSUB *KANSET1
1710   L=264:M=5:S=1:XX=364
1720   GOSUB *XLINE
1730   GOSUB *LINEB1
1740  RETURN
1750  '
1760  *TATAMI2
1770   POINT(100,30):CLS
1780   LINE-STEP(176,44),7,B ←―― 6帖のたたみを描く
1790   LINE STEP(-176,0)-STEP(88,88),7,B
1800   LINE-STEP(176,-44),7,B
1810   LINE STEP(0,44)-STEP(88,-88),7,B
1820   LINE-STEP(-176,-44),7,B
1830   GOSUB *KANSET1
1840   L=352:M=7:S=2:XX=452
1850   GOSUB *XLINE
1860   GOSUB *LINEB2
1870  RETURN
1880  '
1890  *CURSORINIT (Gカーソルのイニシャライズ)
```

(Note for lines 1460–1530: 四角を描いた後に Gカーソルをx，y方向に1Dotずらす (物と物が重ならないようにしている))

```
1900    SCREEN 1,2
1910    VIEW(0,0)-(55,25)
1920    LINE(0,10)-(50,10),7 } Gカーソルを作る
1930    LINE(25,0)-(25,20),7 }
1940    GET (17,6)-(32,14),CU% ← Gカーソルのパターンを読み込む
1950    CLS 2:SCREEN 1,0
1960 '
1970    RESTORE *CDATA
1980    FOR I=0 TO 5
1990      READ INCX(I)（X軸方向）   }
2000    NEXT                        } Gカーソルのステップ数を定義する
2010    FOR I=0 TO 5                }
2020      READ INCY(I)（Y軸方向）   }
2030    NEXT
2040 RETURN
2050 '
2060 *CDATA（Gカーソルのステップ数のデータ）
2070 DATA 1,10,20,30,40,50
2080 DATA 1,5,10,15,20,25 〔白黒200mode〕→解像度2：1
2090 '
2100 *MOVECUR
2110    GOSUB *MESSAGE
2120    LOCATE,,0:CONSOLE 0,25:MC=1
2130    PUT (X-8,Y-4),CU%,XOR ←── Gカーソルが現れる
2140    GOSUB *GETKEY
2150    *LOOP2:IF CH=13 THEN *EXIT2
2160      BEEP 1:BEEP 0
2170      IF CH>47 AND CH=<53 THEN DX=CH-48: } Gカーソルのステップ数を変える
2180         INCX=INCX(DX):INCY=INCY(DX)     }          （数字0～5）
2190      LOCATE 55,21:PRINT USING "Cursor step  ##";INCX
2200      COLOR@(68,21)-(69,21),6
2210      LOCATE 71,21:PRINT "(cm)"
2220      PUT (X-8,Y-4),CU%,XOR ←── Gカーソルが消える }
2230      IF CH=29 THEN X=X-INCX                     }
2240      IF CH=28 THEN X=X+INCX                     }
2250      IF CH=30 THEN Y=Y-INCY                     } Gカーソルの移動を行う
2260      IF CH=31 THEN Y=Y+INCY                     }
2270      GOSUB *LIMIT                               }
2280      PUT (X-8,Y-4),CU%    ←── Gカーソルが現れる }
2290      GOSUB *GETKEY
2300    GOTO *LOOP2
2310    *EXIT2:PUT (X-8,Y-4),CU%,XOR ← Gカーソルが消える
2320    LOCATE,,1
2330 RETURN
2340 '
2350 *GETKEY
2360    CH=ASC(INPUT$(1)) ← キー入力
2370 RETURN
2380 '
2390 *KANSET1                                      }
2400    IF A=1 THEN RESTORE *KDATA1:X=176:F=5      }
2410    IF A=2 THEN RESTORE *KDATA2:X=232:F=3      } "4.5帖"or"6帖"の表示
2420    FOR I=1 TO F                               } （注意）
2430      READ A$:KCODE1=VAL("&h"+A$)              } 漢字ROMを使用していない方は
2440      PUT (X,0),KANJI(KCODE1),PSET,7,0         } このサブルーチンは必要ありません
2450      X=X+16                                   }
2460    NEXT                                       }
2470 RETURN                                        }
2480 '
2490 *KDATA1
2500 DATA 2334,2125,2335,2120,4421
2510 '
2520 *KDATA2
2530 DATA 2336,2120,4421
2540 '
2550 *XLINE
2560    SCREEN 1,0,2,1 ←── メジャーを第3画面に描く
2570    POINT (100,24)                   }
2580    LINE-STEP(L,0),7                 }
2590    POINT STEP(-L,0)                 } X軸方向にメジャーを描く
2600    FOR I=1 TO M                     }
2610      LINE STEP(49,-4)-STEP(0,4),7   }
2620    NEXT                             }
```

```
2630 '
2640 *YLINE
2650   POINT (88,30)
2660   LINE-STEP(0,132),7
2670   POINT STEP(0,-132)                       Y軸方向にメジャーを描く
2680   FOR I=1 TO 5
2690     LINE STEP(-4,24)-STEP(4,0),7
2700   NEXT
2710 RETURN
2720 '
2730 *DISP1
2740   LOCATE 24,2:PRINT"1 :LOCATE 36,2:PRINT"2"
2750   LOCATE 45,2:PRINT"m"
2760   LOCATE 10,9:PRINT"1":LOCATE 10,15:PRINT"2"
2770   LOCATE 10,20:PRINT"m":LOCATE 0,0
2780   COLOR@(10,2)-(46,2),5
2790   COLOR@(10,5)-(10,20),5
2800 RETURN
2810 '
2820 *DISP2
2830   LOCATE 24,2:PRINT"1":LOCATE 36,2:PRINT"2"
2840   LOCATE 49,2:PRINT"3":LOCATE 57,2:PRINT"m"
2850   LOCATE 10,9:PRINT"1":LOCATE 10,15:PRINT"2"
2860   LOCATE 10,20:PRINT"m":LOCATE 0,0
2870   COLOR@(10,2)-(58,2),5
2880   COLOR@(10,5)-(10,20),5
2890 RETURN
2900 '
2910 *MESSAGE
2920   LOCATE 63,0:PRINT"0 --- ( 1cm)"
2930   LOCATE 63,1:PRINT"1 --- (10cm)"
2940   LOCATE 63,2:PRINT"2 --- (20cm)"
2950   LOCATE 63,3:PRINT"3 --- (30cm)"
2960   LOCATE 63,4:PRINT"4 --- (40cm)"
2970   LOCATE 63,5:PRINT"5 --- (50cm)"
2980   COLOR@(63,0)-(63,5),3
2990   COLOR@(64,0)-(76,5),4
3000 RETURN
3010 '
3020 *BOXINIT
3030   LOCATE 57,10:PRINT "ﾄﾞﾜ ﾉ ﾊﾞｼｮ ﾊ !"
3040   GOSUB *BOX1
3050   LOCATE 57,10:PRINT "ﾏﾄﾞ ﾊ !";SPACE$(15)
3060   GOSUB *BOX1
3070   LOCATE 57,10:PRINT "ﾎｶﾆ ﾏﾄﾞ ｶﾞ ｱﾘﾏｽｶ ? y/n":LOCATE,,0
3080   X$=INPUT$(1)
3090   IF X$="y" THEN GOTO 3050
3100   LOCATE 57,10:PRINT "ｵｼｲﾚ ﾊ !";SPACE$(14)
3110   GOSUB *BOX1
3120 RETURN
3130 '
3140 *BOX1
3150   GOSUB *MOVECUR
3160   X1=X:Y1=Y
3170   GOSUB *MOVECUR
3180   IF X=X1 AND Y=Y1 THEN RETURN←ドア，窓，押し入れを描く
3190   IF X=100 AND Y<>Y1 THEN LINE (X1-4,Y1)-(X,Y),7,BF
3200   IF (X=364 OR X=452) AND Y<>Y1 THEN LINE (X1,Y1)-(X+4,Y),7,BF
3210   IF Y=30 THEN LINE (X1,Y1)-(X,Y-2),7,BF
3220   IF Y=162 THEN LINE (X1,Y1)-(X,Y+2),7,BF
3230 RETURN
3240 '
3250 *DELETE1 (dコマンド)
3260   CLS:COLOR@(25,23)-(50,23),3
3270   SCREEN 1,0,1,6
3280   ON S GOSUB *DISP1,*DISP2
3290   GOSUB *MOVECUR
3300   X2=X:Y2=Y
3310   GOSUB *MOVECUR             置いた物を消す
3320   LINE (X2,Y2)-(X,Y),0,BF
3330   ON S GOSUB *LINEB1,*LINEB2
3340 RETURN
3350 '
```

```
3360 *SELECT1
3370   LOCATE 26,22:PRINT "--- Command ---"
3380   COLOR@(26,22)-(40,22),6
3390   LOCATE 5,23:PRINT "move & box (m)";SPACE$(10);
3400   PRINT "delete      (d)          cls      (c)"
3410   LOCATE 5,24:PRINT "copy          (h)";SPACE$(10);
3420   PRINT "end         (stop)";
3430   LOCATE,,0
3440 RETURN
3450 '
3460 *TILEING
3470   IF (N MOD 16)=0 OR C=1 THEN RESTORE *TILEDATA:C=0:N=0
3480   READ A$,B$,C$,D$
3490   TILE$=CHR$(VAL("&H"+A$))+CHR$(VAL("&H"+B$))
                 +CHR$(VAL("&H"+C$))+CHR$(VAL("&H"+D$))
3500   PAINT ((X0+X)/2,(Y0+Y)/2),TILE$,7
3510 RETURN                          物と物を区別するために
3520 '                               異ったタイリングを行う
3530 *TILEDATA
3540 DATA ff,ff,ff,ff,ff,ff,00,00,f0,0f,f0,0f,f0,f0,f0,f0
3550 DATA 55,aa,55,aa,ff,00,ff,00,88,88,88,88,11,22,44,88  16個のタイル
3560 DATA 88,44,22,44,88,22,44,11,88,88,22,22,88,55,22,00  パターンのデータ
3570 DATA c0,c0,00,00,e0,c0,80,00,88,22,88,22,99,66,99,66
3580 '
3590 *LINEB1
3600   SCREEN 1,0,2,1                第3画面に4.5帖の
3610   POINT (100,30)                  部屋の枠を描く
3620   LINE-STEP(264,132),,B
3630 RETURN
3640 '
3650 *LINEB2
3660   SCREEN 1,0,2,1
3670   POINT (100,30)                6帖の部屋の枠を描く
3680   LINE-STEP(352,132),,B
3690 RETURN
3700 '
3710 *ERRTRAP
3720   RESUME NEXT ←── エラー・トラップを行う
3730 '
3740 *SCLS (Cコマンド)
3750   SCREEN 1,0,1,6:COLOR@(52,23)-(65,23),3
3760   IF S=1 THEN SX=363(4.5帖の場合)
3770   IF S=2 THEN SX=451(6帖の場合)
3780   LINE (101,31)-(SX,161),0,BF:C=1 ←── 画面上のすべての物を消す
3790 RETURN
3800 '
3810 *LIMIT
3820   IF X<100 THEN X=100
3830   IF X>XX THEN X=XX
3840   IF Y<30 THEN Y=30
3850   IF Y>162 THEN Y=162
3860 RETURN
3870 '
3880 *GCOPY(hコマンド)
3890   COLOR@(5,24)-(20,24),3
3900   CLS:PRINT "Prepare your printer !!"
3910   PRINT "       Press any key !!"
3920   CLS:PRINT "Copying now !!"
3930   IF INKEY$="" THEN 3930
3940   COPY 2:CLS
3950 RETURN
3960 '
3970 *ABORT
3980   IF MC THEN RETURN ←── Gカーソルが画面上に出て
3990   CONSOLE 0,25:CLS      いる時には受けつけない
4000   LOCATE 0,0,1
4010   STOP OFF
4020 END
```

6.1.4 Roomの使い方

では，このプログラムの使い方について説明しておきます．コマンドは4種類あり，各コマンドは以下の通りです．

○m (move & box)

十字カーソルの移動と物（箱）の位置指定を行います．十字カーソルの移動は，カーソル・キーを用い，方向は，カーソル・キーに準拠します．また，STEP数（移動量）は，テン・キーの0～5で行い，0→1cm，1→10cm，2→20cm，3→30cm，4→40cm，5→50cmに対応しています．物は，十字カーソルで2点を指定すると自動的に描きます．位置指定（座標入力）は，リターン・キーにより行います．

○d (delete)

部屋に置いた物の一部を消去します．消去の仕方は，mコマンドと同様で，十字カーソルによる2点指定です．

○h (hard copy)

画面のコピーを行います．コピーされるのは，第1画面と第2画面の合成画面です．

○c (clear)

部屋に置いた物が，すべて消去されます．

○stop キー (end)

このキーを押すと，プログラム実行が終了します．但し，このキーは，コマンド待ちの時にしか受け付けられません．

まず，プログラムを実行させると，部屋のサイズ（4.5か6）を聞いてきますので，いずれかを入力してください．次に，ドアなどの位置を聞いてきますので，十字カーソルで入力してください．これも2点指定です（図7）．もし存在しない場合は，同じ場所でリターン・キーを2度押してください．以上が終了するとコマンド待ちになります．後は，先程説明したコマンドを入力して部屋のレイアウトを行います．物を置く時には，十字カーソルのステップ数を調整し，横にあるメジャーを参考にしながら入力してください（図8）．

今回のプログラムでは，部屋のサイズが固定であるため汎用的であるといえません．皆さんもこのプログラムを参考に自分の部屋用に作り変えてみてはいかがでしょうか．

部屋の枠の上
でしか受け付けない
第1点
第2点

図7 ドアの位置指定

第1点
第2点
第1点
第2点

図8 部屋の家具の指定

6.2 タイリングを使った服地のシミュレーション(Cloth)

6.1では，白黒モードで3画面を使用して，部屋のレイアウトに挑戦してみましたが，今度は，カラー・モードで，タイリング・ペイントを使った色及び模様の調和について考えてみたいと思います．色の調和にも様々な応用が考えられますが，今回は，洋服を題材にして話を進めていきます．

N_{88}-BASICでは，擬似的ではありますが，125色の中間色と自分で思うままのタイリング・パターンが使いこなせます（4章参照）．しかし，これらの色や模様を塗るPAINT命令を使う時に，必ずと言っていい程悩まされるのが，〝境界色〞です．というのは，境界色の存在により色が塗れたり塗れなかったり，また，たとえ塗れたとしても，今回のような場合には，境界色の存在により色と色の組み合わせが見にくくなってしまい，表現力が落ちてしまうからです．

そこで，今回は，境界色をなくしてしまう方法についてお話しします．手軽な方法としては，パレットを変化させることが一番なのですが，今回は，もう少し汎用的な方法を導入しようと思います．これは，私達がよく知っている切り絵の原理を用いるのです．この方法を使うと，境界色をなくすことができ，さらに既にあるパターンの上に新しいパターンを置くことも可能なのです．

6.2.1 GET@とPUT@を用いた切り絵のテクニック

単に切り絵といっても，紙でやるように簡単にはいきません．画面上でこのような作業を実現するには，色々な下準備を必要とします．

まず，同じ様なパターンを2つ用意し，PUT@命令でそのパターンを置いたり，また，そこにできたパターンをGET@命令で拾ったりします．この繰り返しにより切り絵を行います．では具体的にどのようにすれば良いかを説明していきましょう．

今，図9の（a）のような洋服パターンを（b）の領域に置き，（c）のようなものを作るもの

とします．この動作を実行するには，図10の作業1～4の過程を経ることにより可能です．

図10を見てください．まず，図の（a）の様な2つのパターンをGETし，配列の中に収めて

（a）　（b）　（c）

図9

図10　切り絵の原理

おきます．次に，画面上にワーキング・エリアを設けておき，パターンに付けたい色をワーキング・エリアにペイントしておきます．これで準備はOKです．

まず，初めに作業1を行います．作業1は，これから洋服のパターンを置こうとする場所に，白抜きのパターンを PUT@命令で AND を取り配置します．このとき，AND を取ることにより，図9の(c)のようなパターンの形がくり抜けます．次に作業2です．これは，ワーキング・エリアに，もう1つのパターン，つまり黒抜きのパターンを PUT@命令で AND を取り配置します．これで，洋服のパターンに色が付けられます．そして，この色の付いたパターンを GET します．これが作業3にあたります．最後に作業3で得られたパターンを，作業1でくり抜かれた部分に PUT@命令で OR を取り配置します．これで，図10の（d）の様な目的の図形が完成する訳です．

ここで最も重要なポイントは，PUT をする時の〝AND〟と〝OR〟です．〝AND〟を取るということは，パターンの黒抜き（ビットが立っていない）部分は消えてしまい，白抜き（ビットが立っている）部分だけが残る訳です．また，作ったパターンで〝OR〟を取るということは，和をとることを意味しますので，お互いのパターン同志が干渉しあうことなく絵をはめ込むことができる訳です．この説明でよく内容が飲み込めない方は，リスト2を参考にして，実際に実行してみると内容が摑めると思います．この様な PUT の使い方は，割と気が付かないと思います．知らなかった人は，しっかり覚えておいてください．今回の洋服の色の調和を行う場合，この様な手法は非常に効果的です．実は，第1巻のカラーページの最初の写真（付録のプログラム）は，このテクニックを使って作成したものです．

6.2.2 プログラムの説明

このプログラムでは，今説明した洋服の色をはめ込むルーチンが，ほとんどを占めています．しかし，この切り絵の原理も，色々な洋服のパターン（今回は，ブラウス，スカート，ズボン，ジャケットを用意しました）を使用することになると，カラー・モードでは多少問題が生じてきます．というのは，大きなパターンになるとかなり大きな配列を取らなければならないからです．それも，それぞれについて2つ分もです．これでは，配列がいくらあっても足りません．

そこで今回は，カラー・モードと白黒モードをうまく使うことにしました．リスト2の＊UCLOTHINIT と＊UCLOTH を見てください．服の2つのパターンを GET で拾う際に，白黒モードにして拾います．白黒モードであると配列の大きさがカラー・モードに比べ，約1／3で済みます．また服のパターンを PUT で置く時にも，やはり白黒モードで行います．但し，ここで注意しなければならないことは，この服のパターンを第1画面から第3画面まで全部に，ページを切り換えながら置いていくということです．こうすることにより，カラー・モードに復帰した時に，元通りの色に再現できます．このことは，白黒3ページ・モードの各ページが，カラー・モードの時の各色バンクと物理的には同じであるということを利用しているのです．

後は，カラー・モードに切り換えて，できた服のパターンを GET して，自分の思う所に PUT

(白黒モード)　　　　　　　　　　　　(カラー・モード)

ワーキング・エリア

〔第1画面〕　〔第2画面〕　〔第3画面〕

PUT AND　　PUT AND　　GET　PUT OR

図11　モードの切り換え

でおくだけです．この時 GET するパターンはカラー・モードですから，それなりの配列を取ってあります（図11）．

それでは次に，リスト2の全体的な説明をしておきましょう．今回の服のパターンは，汎用性を考えて，すべてスクリーン・レイアウト用紙から座標データを拾い，LINE 文で書いてありますが，もしこのような方法が面倒だと思う方は，4章で紹介したGエディタなどを使ってパターンを作ることも可能です．

注意：このプログラムには，4章の中間色ジェネレータとタイル・パターン・ジェネレータが必要です．

リスト2

```
1000 '
1010 ' >>> Cloth  (sample)   [ (640 x 200) mode ] <<<
1020 '
1030 '
1040 '
1050 ON ERROR GOTO *ERRTRAP
1060 '
1070 ON STOP GOSUB *ABORT
1080 STOP ON
1090
1100 SCREEN 0,3:CLS 3:WIDTH 80,25:CONSOLE 0,25,0,1
1110 COLOR 7,0,0,7:SCREEN ,0
1120 FOR I=0 TO 7:COLOR=(I,I):NEXT ← パレットを初期状態にする
1130 UX=0:UY=0:DX=0:DY=0:TX=0:TY=0:UX1=0:UY1=0:XM=10:YM=130
1140 DIM W%(1596),U%(419),U1%(419),UU%(534),UU1%(534)
1150 DIM D%(369),D1%(369),P%(331),P1%(331),TILE.G$(9)
1160 GOSUB *MID.INIT    }
1170 GOSUB *MID.DISP    } 4章のツール
1180 GOSUB *TILE.INIT   }
1190 GOSUB *MTILEDISP   パターンの読み込みは，すべて
1200 SCREEN 1,0:GOSUB *UCLOTHINIT   白黒モードで行う
1210 GOSUB *UCLOTH1INIT
1220 GOSUB *DCLOTHINIT
1230 GOSUB *PANTALOONINIT
1240 GOSUB *COMMAND
1250 SCREEN 0,0 ← カラー・モードにする
```

W%…ワーキング・エリア
U%…ブラウスの黒ぬきパターン
U1%…　〃　白　〃
UU%…ジャケットの黒ぬきパターン
UU1%…　〃　白　〃
D%…スカートの黒ぬきパターン
D1%…　〃　白　〃
P%…ズボンの黒ぬきパターン
P1%…　〃　白　〃
TILE.G%…タイル・パターン

```
1260 LINE (230,40)-(410,190),3,B
1270 '
1280 *LOOP
1290   CONSOLE 6,18:LOCATE,,0:PRINT
1300   A$=INPUT$(1)
1310   IF A$="b" THEN GOSUB *UCLOTH
1320   IF A$="j" THEN GOSUB *UCLOTH1
1330   IF A$="s" THEN GOSUB *DCLOTH
1340   IF A$="p" THEN GOSUB *PANTALOON
1350   IF A$="c" THEN GOSUB *DISPCLS
1360   IF A$="m" THEN GOSUB *MAKETILE
1370 GOTO *LOOP
1380 '
1390 *UCLOTHINIT ←── ブラウスのパターン作成
1400   POINT (UX,UY)
1410   LINE STEP(30,6)-STEP(24,-4),7
1420   RESTORE *DATA1
1430   T=8:GOSUB *XYLINE
1440   LINE STEP(56,0)-STEP(16,24),7
1450   LINE-STEP(16,-4),7
1460   LINE-STEP(-24,-36),7
1470   CIRCLE STEP(-36,-4),12,7,3.14,0,.3
1480   LINE STEP(-66,-2)-STEP(130,48),7,B
1490   PAINT (UX+2,UY+2),7,7
1500   GET(0,0)-(130,48),U%  ←── 黒ぬきのパターンを読み込む ┐
1510   PUT (0,0),U%,PRESET   ←── パターンの反転            ├ ※
1520   GET (0,0)-(130,48),U1% ←── 白ぬきのパターンを読み込む ┘
1530   GOSUB *WCLR
1540 RETURN
1550 '
1560 *DATA1
1570 DATA 24,0,24,4,-16,36,-40,0
1580 DATA -16,-36,-24,36,16,4,16,-24
1590 '
1600 *DCLOTHINIT←── スカートのパターン作成
1610   POINT (DX,DY)
1620   LINE STEP(26,2)-STEP(-20,56),7
1630   LINE-STEP(80,0),7
1640   LINE-STEP(-20,-56),7
1650   LINE-STEP(-40,0),7
1660   LINE STEP(-26,-2)-STEP(90,60),7,B
1670   PAINT (DX+2,DY+2),7,7
1680   GET(0,0)-(90,60),D%    ┐
1690   PUT (0,0),D%,PRESET    ├ ※印参照
1700   GET (0,0)-(90,60),D1%  ┘
1710   GOSUB *WCLR
1720 RETURN
1730 '
1740 *PANTALOONINIT←── ズボンのパターン作成
1750   POINT (TX,TY)
1760   RESTORE *DATA2
1770   LINE STEP(12,2)-STEP(-6,8)
1780   T=8:GOSUB *XYLINE
1790   LINE STEP(-10,-2)-STEP(65,72),7,B
1800   PAINT (TX+2,TY+2),7,7
1810   GET(0,0)-(65,72),P%    ┐
1820   PUT (0,0),P%,PRESET    ├ ※印参照
1830   GET (0,0)-(65,72),P1%  ┘
1840   GOSUB *WCLR
1850 RETURN
1860 '
1870 *DATA2
1880 DATA 6,60,18,0,2,-52,2,52
1890 DATA 16,0,6,-60,-6,-8,-40,0
1900 '
1910 *UCLOTH1INIT ←── ジャケットのパターン作成
1920   POINT(UX1,UY1)
1930   RESTORE *DATA3
1940   LINE STEP(60,6)-STEP(-24,4),7
1950   T=4:GOSUB *XYLINE
1960   CIRCLE STEP(0,-4),8,7,4.71,0
1970   LINE STEP(8,0)-STEP(0,-42)
1980   T=7:GOSUB *XYLINE
```

初期画面

```
1990   CIRCLE STEP(0,-4),8,7,3.14,4.71
2000   LINE STEP(-8,0)-STEP(0,-42),7
2010   T=8:GOSUB *XYLINE
2020   LINE STEP(56,0)-STEP(19,27),7
2030   LINE-STEP(16,-4),7
2040   LINE-STEP(-27,-40)
2050   LINE STEP(-108,-10)-STEP(143,58),7,B
2060   PAINT (UX1+2,UY1+2),7,7
2070   GET(0,0)-(143,58),UU%         ┐
2080   PUT (0,0),UU%,PRESET          ├※印参照
2090   GET (0,0)-(143,58),UU1%       ┘
2100   GOSUB *WCLR
2110 RETURN
2120 '
2130 *XYLINE
2140   FOR I=1 TO T
2150     READ X,Y:LINE-STEP(X,Y),7
2160   NEXT
2170 RETURN
2180 '
2190 *DATA3
2200 DATA 8,18,2,16,-2,12,14,0
2210 DATA -6,-4,24,0,24,4,-8,18,-2,16,2,12,-14,0
2220 DATA 6,-4,0,-2,-24,0,0,2,-24,4,-27,40,16,4,19,-27
2230 '
2240 *WCLR ←── ワーキング・エリアのクリアを行う
2250   LINE (0,0)-(150,80),0,BF
2260 RETURN
2270 '
2280 *MTILEDISP
2290   RESTORE *DATA4
2300   FOR I=0 TO 9
2310     READ TL:TILE$=""
2320       FOR J=1 TO TL
2330         READ B$,R$,G$
2340         TILE$=TILE$+CHR$(VAL("&H"+B$))+CHR$(VAL("&H"+R$))+CHR$(VA
L("&H"+G$))
2350       NEXT
2360     LINE (XM,YM)-(XM+26,YM+26),7,B
2370     PAINT (XM+2,YM+2),TILE$,7
2380     TILE.G$(I)=TILE$ ←── タイル・ストリングを読み込む
2390     XM=XM+32
2400     IF XM>140 THEN XM=10:YM=YM+31
2410   NEXT
2420 RETURN
2430 '                                                  10個のタイル・パターン
2440 *DATA4 ↙FOR～NEXTループを回す回数                    のデータ
2450 DATA [2],ff,00,ff,00,ff,00,2,00,ff,ff,ff,00,00          ┐
2460 DATA [2],ff,ff,ff,ff,00,00,2,00,00,ff,00,ff,00          │
2470 DATA [3],ff,e7,ff,ff,00,ff,ff,e7,ff                     │
2480 DATA [3],00,ff,e7,00,ff,00,00,ff,e7                     │
2490 DATA [4],00,ff,ff,00,ff,ff,ff,00,00,ff,00,00            │
2500 DATA [4],e7,ff,e7,00,ff,00,00,ff,00,e7,ff,e7            │
2510 DATA [4],ff,00,ff,ff,00,ff,ff,ff,00,ff,ff,00            │
2520 DATA [6],e7,ff,e7,c3,ff,c3,81,ff,81,c3,ff,c3,e7,ff,e7,ff,ff,ff ┘
2530 '
2540 *UCLOTH ←──── ブラウスのパターンを画面上に置く
2550   COLOR@(23,2)-(38,2),3
2560   GOSUB *SELECT
2570   FOR P=0 TO 2
2580     SCREEN 1,0,P,2^P       ←白黒モードで行う        ┐
2590     PUT (255,60),U%,AND ←─(図の作業1)                │
2600     PUT (0,0),U1%,AND      ←(図の作業2)             │
2610   NEXT                                               │
2620   SCREEN 0,0               ←カラーモードへ           ├※※
2630   GET (0,0)-(130,48),W% ←(図の作業3)                 │
2640   GOSUB *WCLR                                        │
2650   PUT (255,60),W%,OR       ←(図の作業4)             │
2660   COLOR@(23,2)-(38,2),7                              ┘
2670 RETURN
2680 '
2690 *UCLOTH1 ←── ジャケットのパターンを画面上に置く
2700   COLOR@(38,3)-(60,3),3
```

```
2710    GOSUB *SELECT
2720    FOR P=0 TO 2
2730      SCREEN 1,0,P,2^P
2740      PUT (249,56),UU%,AND
2750      PUT (0,0),UU1%,AND
2760    NEXT
2770    SCREEN 0,0                    ※※印参照
2780    GET (0,0)-(143,58),W%
2790    GOSUB *WCLR
2800    PUT (249,56),W%,OR
2810    COLOR@(38,3)-(60,3),7
2820  RETURN
2830  '
2840  *DCLOTH ←── スカートのパターンを置く
2850    COLOR@(38,2)-(60,2),3
2860    GOSUB *SELECT
2870    FOR P=0 TO 2
2880      SCREEN 1,0,P,2^P
2890      PUT (276,99),D%,AND
2900      PUT (0,0),D1%,AND
2910    NEXT
2920    SCREEN 0,0                    ※※印参照
2930    GET (0,0)-(90,60),W%
2940    GOSUB *WCLR
2950    PUT (276,99),W%,OR
2960    COLOR@(38,2)-(60,2),7
2970  RETURN
2980  '
2990  *PANTALOON ←──── ズボンのパターンを置く
3000    COLOR@(23,3)-(38,3),3
3010    GOSUB *SELECT
3020    FOR P=0 TO 2
3030      SCREEN 1,0,P,2^P
3040      PUT (290,99),P%,AND
3050      PUT (0,0),P1%,AND
3060    NEXT
3070    SCREEN 0,0                    ※※印参照
3080    GET (0,0)-(65,72),W%
3090    GOSUB *WCLR
3100    PUT (290,99),W%,OR
3110    COLOR@(23,3)-(38,3),7
3120  RETURN
3130  '
3140  *PBOX ←── ワーキング・エリアに選ばれた中間色を
                ペイントする
3150    LOCATE,,1:LINE (0,0)-(143,72),7,B
3160    GOSUB *MID.SEL
3170    PAINT (1,1),TILE.M$,7
3180  RETURN
3190  '
3200  *DISPCLS ←── 置かれた服のパターンをすべて消す
3210    COLOR@(23,4)-(38,4),3
3220    LINE (231,41)-(409,189),0,BF
3230    COLOR@(23,4)-(38,4),7
3240  RETURN
3250  '
3260  *COMMAND
3270    LOCATE 30,0:PRINT ">>>>> Command <<<<<"
3280    COLOR@(30,0)-(50,0),6
3290    LOCATE 23,2:PRINT "Blouse      (b)     Skirt      (s)"
3300    LOCATE 23,3:PRINT "Pantaloon   (p)     Jacket     (j)"
3310    LOCATE 23,4:PRINT "Clear       (c)     Maketile   (m)"
3320  RETURN
3330  '
3340  *NUMBER
3350    LOCATE 3,17:PRINT"0    1    2    3    4"
3360    LOCATE 3,21:PRINT"5    6    7    8    9"
3370  RETURN
3380  '
3390  *ERRTRAP
3400    RESUME NEXT ←── エラー・トラップ
3410  '
3420  *ABORT
3430    CONSOLE 0,25:CLS
```

タイル・パターンを作る

タイル・パターンのデータ

```
3440    SCREEN 0,0:LOCATE,,1
3450    STOP OFF
3460 END
3470 '
3480 *MAKETILE
3490    CLS:GOSUB *TILEGEN
3500    GOSUB *NUMBER
3510    LOCATE 0,10:INPUT "Define number (0-9)";N
3520    TILE.G$(N)=TILE.T$ ←── 作ったパターンのタイル・ストリングを読み込む(再定義)
3530    IF 0=<N AND N=<4 THEN LINE (11+32*N,131)-STEP(24,24),0,BF:PAINT
 (13+32*N,133),TILE.T$,7
3540    IF 5=<N AND N=<9 THEN LINE (11+32*(N-5),162)-STEP(24,24),0,BF:P
AINT (13+32*(N-5),164),TILE.T$,7
3550    GOSUB *DATA.DISP
3560    LOCATE 0,11:PRINT "Retry  ?";
3570    CH.G=ASC(INPUT$(1))
3580    IF CH.G=&H6E THEN CLS:GOSUB *COMMAND:RETURN
3590 GOTO *MAKETILE
3600 '
3610 *SELTILE
3620    CLS:LOCATE,,1:GOSUB *NUMBER
3630    LOCATE 0,13:INPUT "Pattern number (0-9)";NN
3640    IF NN<0 OR NN>9 THEN GOTO *SELTILE
3650    LINE (0,0)-(143,72),7,B
3660    PAINT (1,1),TILE.G$(NN),7:CLS ←── ワーキング・エリアに選ばれたタイル・パターンを
                                          ペイントする
3670 RETURN
3680 '
3690 *SELECT
3700    LOCATE 0,10:PRINT "Tiling --------> CR"
3710    PRINT "Midde color --->SPACE"
3720    CH.G=ASC(INPUT$(1)):CLS
3730    IF CH.G=&HD THEN GOSUB *SELTILE
3740    IF CH.G=&H20 THEN GOSUB *PBOX
3750 RETURN
```

(3530–3540 注: 表示されている10個の模様の1つを作ったものに換える)

〈中間色ジェネレータ入る〉
〈タイル・パターン・ジェネレータ入る〉

6.2.3 Clothの使い方

今回服の種類は4つ用意してあり，これらの組み合わせにより色の調和を見る様になっています．コマンドは6種類あり，機能は以下の通りです．

○b (blouse)

このコマンドは，画面上にブラウスのパターンを配置します．このコマンドに入ると，中間色（125色）の中から選ぶか，タイル・パターンを選ぶかを聞いてきますので，中間色の場合は〝スペース・キー〞をタイル・パターンの場合は〝リターン・キー〞を入力します．中間色の選択の仕方は，4章で紹介した中間色ジェネレータの説明を参照してください．タイル・パターンの選択は，画面上左側の10個のパターンのいずれかを選び，0～9の数字を入力します．このいずれかの操作が終了すると，選ばれた色のブラウスが画面上に表れます．

○s (skirt)　○p (pantaloon)　○j (jacket)

これらの3つのコマンドは，それぞれスカート，ズボン，ジャケットのパターンを配置するものです．これらのコマンドの使用法は，b (blouse) に準拠します．

○c (clear)

すでに画面上に置かれた服のパターンをすべて消去します．このコマンドは新しくやり直す場合に，または間違った操作をした場合に使用します．

○m (make tile)

このコマンドは，すでに定義されている10個のタイル・パターンを新しく定義するものです．タイルの作り方については，4章のタイル・パターン・ジェネレータを参照してください．ここで作ったタイル・パターンは，既に定義されている中で，交換したいタイル・パターンの数字を入力することで，再定義されます．また，このタイル・パターンをプログラムの初期状態において出力させたい場合は，最後に出力されるタイルのデータを書き留めておき，*DATA 4のタイル・データを書き換えておいてください．これで次からのプログラムの実行では，自分の好きなタイル・パターンが最初から出力されるようになります．

このプログラム（リスト2）を実行させますと，イニシャライズ(服のパターンを配列にしまっておく）を行った後に，すぐコマンド待ちとなります．ここで先程説明したコマンドを用いて服を組み合せていきます．但し，気を付けなければならないことは，ジャケットは常に最後に置かなければならないことです．これは，切り絵の原理を応用しているためいたしかたありません．

今回のプログラムに，もう少し手を加えると完全な着せ替えができると思います．例えば，人の絵を用意（Gエディタより作る）し，これらの服の他に，帽子や靴，あるいはスカーフなどを加えるとかなり見ばえのするものができることでしょう．また，男性版を作り，背広とネクタイとの色の調和などもおもしろい題材でしょう．但し，このようなパターンが増えてくると配列が足りなくなってきますから，ディスクやテープなどの外部記憶装置を利用することになるでしょう．

6.3 LINE文を主体とした作画(Car)

今回は，Gエディタや特殊なテクニックを用いずに，スクリーン・レイアウト用紙(APPENDIXに掲載）に作画し，そのデータによりLINE文を主体として作ったプログラムを紹介します．内容は，背景として，都会的なものと田舎的なものを用意し，それぞれに車を描き，それぞれの画面で色の対応を見ることができます．

6.3.1 LINE文による作画の長所と短所

LINE文で描く場合，どうしても幾何学的な絵になりがちです．ビルディングなどの直線的な風景であれば，あまり問題はないのですが，山や木の様な曲線や細かい部分が多くなってくると，なかなかうまく描けません．この様な場合は，Gエディタなどのツールを使用した方が，うまく描けます．

この様に，曲線的，あるいは細かい絵の場合は，データを拾うのが大変であるため不向きです．しかし，LINE 文でなければ，できない芸当もあります．それは，絵の拡大・縮小の問題です．リスト 3の＊BACKGROUND 1， 2を見て頂ければ分かると思いますが，LINE 文で描いておくと，N_{88}-BASICの最たる特徴であるウィンドウとビューポートの活用が可能なのです．Gエディタなどの様にGET／PUT 文を数多く使用するプログラムでは，ウィンドウを導入することができません．従って，LINE 文だけで描いた絵では，大きな融通性が生まれてくる訳なのです．今回の様に，背景の大きさが異なる場合には非常に効果的です．

次に，描いた絵の再現性について考えてみましょう．Gエディタで描いた絵を保存する場合，必ず全データをディスクやテープに落さなければなりません．ディスクを持っている方は，あまり時間的問題はありませんが，テープの方はかなり大変です．さらに膨大なデータ量となります．しかし，LINE 命令で描いた場合，データを拾うのはかなり労力を必要としますが，データができてしまえばデータが始点座標と終点座標でよいため実行速度も速いし，確実に再現されます．

以上，色々な事を述べてきましたが，絵を描く場合には，条件によってLINE 命令のみを使うことも充分効果があるということです．

プログラムについては，ほとんどがLINE とPOINT 文ですから，特に説明の必要はないでしょう．但し，色の効果を強くするためパレットに少し手を加えています．

リスト3

```
1000 '
1010 '  >>> car (sample) <<<
1020 '
1030 '
1040 '
1050 SCREEN 0,3:CLS 3:SCREEN,0:COLOR 7,0,0,7
1060 CONSOLE 0,25,0,1:WIDTH 80,25:DX1=30:C=5
1070 FOR I=0 TO 7:COLOR=(I,I):NEXT←―― パレットを初期状態にする
1080 '
1090 *MAIN
1100   PRINT "Background"
1110   INPUT "    City (1) or Country (2)";B
1120   CLS:ON B GOSUB *BACKGROUND1,*BACKGROUND2
1130   IF B<>1 AND B<>2 THEN *MAIN
1140 *LOOP
1150   CLS:X1=13:DX=9:GOSUB *XCOLOR
1160   LOCATE 0,4,0
1170   PRINT "color (0-7) or change[background] (c)"
1180   PRINT "      or palette (p) or end (e) ?";
1190   CH=ASC(INPUT$(1))
1200   IF CH=99 THEN *RESTART
1210   IF CH=101 THEN *XEND
1220   IF CH=112 THEN GOSUB *PALET
1230   IF CH<48 OR CH>55 THEN *LOOP
1240   C=CH-48:CLS:GOSUB *CAR
1250 GOTO *LOOP
1260 '
1270 *BACKGROUND1
1280   GOSUB *CITY
1290   WINDOW (0,0)-(639,199)  }車の大きさを変える(縮小)
1300   VIEW (150,60)-(638,198) }
1310   GOSUB *CAR
1320 RETURN
1330 '
1340 *BACKGROUND2
1350   GOSUB *COUNTRY
```

```
1360    WINDOW (0,0)-(639,199)  } 車の大きさを変える(縮小)
1370    VIEW (400,115)-(479,139)}
1380    GOSUB *CAR
1390 RETURN
1400 '
1410 *CAR (車の線を描く)
1420    POINT(150,100)
1430    CIRCLE STEP(105,80),35,C
1440    PAINT STEP(0,0),0,C
1450    CIRCLE STEP(0,0),23,C
1460    TILE$=CHR$(&H88)+CHR$(&H88)+CHR$(&H88)+CHR$(&H22)+CHR$(&H22)+CH
R$(&H22)
1470    PAINT STEP(0,0),TILE$,C
1480    LINE STEP(-35,0)-STEP(-70,0),C
1490    RESTORE *LDATA1:L=9:GOSUB *XYLINE
1500    CIRCLE STEP(-35,0),35,C:PAINT STEP(0,0),0,C
1510    CIRCLE STEP(0,0),23,C:PAINT STEP(0,0),TILE$,C
1520    LINE STEP(-35,0)-STEP(-175,0),C
1530    LINE STEP(-10,-40)-STEP(55,-27),C
1540    LINE-STEP(133,0),C:LINE-STEP(86,28),C
1550    LINE-STEP(-276,0),C:PAINT STEP(0,10),C,C
1560    LINE (430,110)-(432,140),C,BF
1570    CIRCLE (190,140),5,C,,,1.3
1580    PAINT (190,140),C,C
1590 RETURN
1600 '
1610 *LDATA1
1620 DATA -10,-10,10,-20,120,-10,60,-30,140,0
1630 DATA 120,40,0,20,-20,10,-35,0
1640 '
1650 *CITY (町の絵(背景)を描く)
1660    VIEW (0,0)-(639,199)
1670    LINE STEP(0,190)-STEP(639,0),7
1680    LINE STEP(0,9)-STEP(-639,0),7
1690    LINE STEP(30,-199)-STEP(0,190),7
1700    LINE STEP(50,-180)-STEP(100,40),7,B  }
1710    LINE STEP(10,0)-STEP(100,-40),7,B    }
1720    LINE STEP(10,0)-STEP(100,40),7,B     } 建物の窓を描く
1730    LINE STEP(10,0)-STEP(100,-40),7,B    }
1740    LINE STEP(10,-5)-STEP(-450,50),7,B   }
1750    LINE STEP(-40,10)-STEP(520,0),7
1760    LINE STEP(0,-65)-STEP(0,190),7
1770    LINE STEP(0,-95)-STEP(-520,0),7
1780    LINE STEP(40,0)-STEP(80,95),7,B      }
1790    LINE STEP(-70,-90)-STEP(60,85),7,B   }
1800    LINE STEP(2,-55)-STEP(6,20),7,BF     } 建物のドアを描く
1810    LINE STEP(4,-55)-STEP(80,95),7,B     }
1820    LINE STEP(-70,-90)-STEP(60,85),7,B   }
1830    LINE STEP(-68,-55)-STEP(6,20),7,BF   }
1840    LINE STEP(110,-55)-STEP(0,95),7,B
1850    LINE STEP(10,-20)-STEP(260,-65),7,B
1860    LINE STEP(40,85)-STEP(0,-190),7
1870    RESTORE *TILEDATA1
1880    FOR I=1 TO 19
1890      READ X,Y,A$,B$,C$,D$,E$,F$
1900      TILE$=CHR$(VAL("&H"+A$))+CHR$(VAL("&H"+B$))+CHR$(VAL("&H"+C$)
)+CHR$(VAL("&H"+D$))+CHR$(VAL("&H"+E$))+CHR$(VAL("&H"+F$))
1910      PAINT (X,Y),TILE$,7 ←── 中間色を用いている
1920    NEXT
1930 RETURN
1940 '
1950 *TILEDATA1
1960 DATA 10,10,0,aa,88,0,55,22,40,10,aa,ff,ee,55,ff,bb           }
1970 DATA 100,8,aa,aa,aa,55,55,55,100,80,88,0,0,22,0,0            }
1980 DATA 40,120,0,ff,88,0,ff,22,100,97,aa,aa,aa,55,55,55         }
1990 DATA 200,97,aa,aa,aa,55,55,55,250,120,0,ff,88,0,ff,22        } ペイントの時
2000 DATA 300,100,aa,ff,ee,55,ff,bb,560,100,0,aa,88,0,55,22       } の座標とタイ
2010 DATA 620,100,88,88,88,22,22,22,100,30,88,ff,ff,22,ff,ff      } ル・パターン
2020 DATA 250,30,88,ff,ff,22,ff,ff,350,30,88,ff,ff,22,ff,ff       } のデータ
2030 DATA 450,30,88,ff,ff,22,ff,ff,100,150,88,ff,ff,22,ff,ff      }
2040 DATA 200,150,88,ff,ff,22,ff,ff,400,150,88,ff,ff,22,ff,ff     }
2050 DATA 100,195,88,88,88,22,22,22
```

```
2060 '
2070 *XCOLOR ←── 車の色（原色）を表示する
2080   LOCATE 3,24:X1=13:DX=9
2090   PRINT "クロ :  0   ";
2100   FOR I=1 TO 7
2110     PRINT "■ : ";I;"   ";
2120     COLOR@(X1,24)-(X1+DX,24),I
2130     X1=X1+9
2140   NEXT
2150 RETURN
2160 '
2170 *COUNTRY（郊外の背景を描く）
2180   VIEW (0,0)-(638,190)
2190   POINT (0,140)
2200   LINE-STEP(639,0),7
2210   LINE STEP(0,-80)-STEP(-50,-10),7
2220   RESTORE *LDATA2:L=11:C=7:GOSUB *XYLINE
2230   LINE STEP(180,0)-STEP(50,-15),7
2240   RESTORE *LDATA3:L=5:C=7:GOSUB *XYLINE      } 山と雲の絵を描く
2250   LINE STEP(-639,-55)-STEP(50,4),7
2260   RESTORE *LDATA4:L=6:C=7:GOSUB *XYLINE
2270   PAINT (10,10),7,7
2280   RESTORE *TILEDATA2
2290   FOR I=1 TO 4
2300     READ X,Y,AA$,BB$,CC$,DD$,EE$,FF$
2310     TILE$=CHR$(VAL("&H"+AA$))+CHR$(VAL("&h"+BB$))+CHR$(VAL("&H"+C
C$))+CHR$(VAL("&H"+DD$))+CHR$(VAL("&H"+EE$))+CHR$(VAL("&H"+FF$))
2320     PAINT (X,Y),TILE$,7 ←── 中間色を用いている
2330   NEXT
2340   FOR I=0 TO 12 ←── 木の絵を描く
2350     CIRCLE (30+DX1*I,110),15,4,,,1.3
2360     PAINT (29+DX1*I,110),4,4
2370     LINE (30+DX1*I-1,125)-(30+DX1*I+2,140),2,BF
2380   NEXT
2390   RESTORE *DATA1
2400   FOR I=1 TO 3
2410   READ XX,YY,R
2420   CIRCLE (XX,YY),R,6,,,.07
2430   PAINT (XX+1,YY),6,6
2440   NEXT
2450 RETURN
2460 '
2470 *TILEDATA2
2480 DATA 160,150,0,88,aa,0,22,55,320,120,0,0,88,0,0,22
2490 DATA 630,130,0,aa,88,0,ff,22,50,100,ff,0,ee,ff,0,bb
2500 '
2510 *DATA1
2520 DATA 100,160,50,300,175,70,500,155,50
2530 '
2540 *LDATA2
2550 DATA -50,-7,-50,-5,-50,-3,-50,0,-40,5,-50,10
2560 DATA -50,12,-50,15,-30,13,-50,22,-40,28
2570 '
2580 *LDATA3                                     } LINE文のデータ
2590 DATA 50,-13,50,-10,50,-8,100,-13,80,-10       (*XYLINE)
2600 '
2610 *LDATA4
2620 DATA 50,2,50,0,50,-3,50,-5,50,-5,80,-10
2630 '
2640 *RESTART
2650   SCREEN 0,3:CLS:PRINT "working !"
2660   CLS 2:SCREEN,0:CLS:LOCATE,,1
2670 GOTO *MAIN
2680 '
2690 *XYLINE
2700   FOR I=1 TO L
2710     READ LX,LY:LINE-STEP(LX,LY),C
2720   NEXT
2730 RETURN
2740 '
2750 *PALET（背景のパレットを変える）
2760   CLS:LOCATE 0,4:PRINT "palette   (0) ---> (1-7)"
```

```
2770   PRINT:PRINT '      Press any key  [ except "q" ]  !!'
2780   PRINT '       [esc] -----> q'
2790   FOR I=1 TO 7
2800     *EXIT:A$=INKEY$
2810     IF A$='' THEN *EXIT
2820     IF A$='q' THEN *ESC
2830     COLOR=(0,I) ←── パレット番号"0"を"1～7"に対応させる
2840   NEXT
2850   *ESC:FOR I=0 TO 7:COLOR=(I,I):NEXT ←── パレットを初期状態にする
2860 RETURN
2870 '
2880 *XEND
2890   CLS
2900   LOCATE,,1
2910 END
```

パレットを変えると色合いが変わる

6.3.2 Carの使い方

プログラムを実行させますと，初めに都会の背景にするか，田舎の背景にするかを聞いてきますので，〝1″か〝2″の数字を入力してください．いずれかの背景を描いた後，コマンド待ちになります．コマンドは，4種類で機能は以下の通りとなります．

○0～7（数字）

指定された数字の色に車を塗り変えます．色は原色（8色）で画面の下に数字との対応が出力されます．

○c（change background）

背景を変えるコマンドです．このコマンドが入力されると初期の状態に戻ります(＊RESTART)．

○p（palette）

パレットを回すコマンドです．パレットは，0番（黒）のみが，1から7まで変化します．〝q″以外のキーを押すと，パレットの数字がインクリメントされます．途中で抜ける時には，〝q″を押してください．

このコマンドの目的は，背景の色合いを対応する色の傾向に近づけるためです．かなり，画面の雰囲気が変わります（例えば黄色にすると，田舎の背景は紅葉した様になる）．

○e (end)

実行を終了するコマンドです.

6.4 白黒パターンのデザイン(Design)

これまでに, 図形や絵などを作るプログラムとして, 〝Gエディタ〟と〝シンボル・ジェネレータ〟を紹介しました. しかし, その特徴を見てみると, Gエディタは, どちらかと言うと大まかな形のものをデザインするのに向いており, シンボル・ジェネレータは, 精密ではあるが, 比較的小さなパターンを作成するのに向いています. その上, これら双方ともカラー・モードにおけるパターンを作成するものでした.

そこで本節では, 白黒モードにおいて, 大きなパターンを比較的精密に描けるプログラムを紹介します. プログラムは, 十字カーソルと各種のコマンド(直線, 円など)を使って作画していく形式であり, ほぼ, 以前に紹介した〝Room〟及び〝Gエディタ〟を踏襲しています. ですから, これらの機能及びプログラミングについては, そちらを参照して頂くことにして, ここでは, このプログラム特有の事柄に話を絞ることにします.

6.4.1 プログラムの説明

☆十字カーソル

十字カーソルの表示の仕方及びその操作方法は, 6.1の〝Room〟とほとんど同じですが, 今回は, 斜方向にも移動可能な様にプログラムしてあります.

☆マス目の表示

パターンや文字などを正確に作成しようとすると, グラフ用紙の様に基準となるマス目が必要になってきます. そこで今回は, それにあたるものとして点線(実線であると十字カーソルが見にくくなってしまうため)で, マス目を用意しました. しかし, このマス目が同一画面上に存在すると, 描いたパターンを消した場合, マス目も一緒に消えてしまうことになります. そこで, 〝Room〟の時と同じ様に, 3画面を切り替えて用いることにしました. 画面の使用の仕方は, マス目を第2画面に描き, デザインする時には, 第1画面をアクティブにしています. また, デザインする画面の領域は, マス目の内側のみであり, 十字カーソルは, マス目の外には出ない様になっています(*MOVECUR参照).

☆作画コマンド

次に, 実際にパターンを作っていくコマンドを説明します. 前記した様に〝Gエディタ〟と重複する部分は, 詳しく説明しません.

ペン先
のパターン

PUT OR

PUT OR
（1ドットずらして置く）

これを連続で用いると
太い線が得られる.

図12 インキングの作り方

まず，線を引くには LINE 命令を，箱を描くには LINE の box を，円を描くには CIRCLE 命令を用いています．但し，CIRCLE においては，円だけでは本プログラムの目的として機能不足のため，方向を 8 方向用意し，半円や扇形を描けるようにしました．これは，初めに角度を定義しておき，それに対応した方向で選べるようにしました．他に，点を打つには PSETを，またこれらを消去するには，パレット番号 0 （黒）にして行っています．また，今回はこれに加え，インキングという機能を設けました．これは，画面上にフリー・ハンドでパターンをデザインする時に使用するもので，ペンの太さが，1と10（ドット）用意してあります．太さが10の場合は，マジックで描いた様な効果が得られるはずです．では，このインキングの作り方について説明しましょう．太さ 1 の方は，単に PSET を連続に用いて行っています．太さ10の方は，多少工夫をしなければなりません．まず．リスト 4 の＊INKINIT を見てください．これは，図12の様なペン先のパターンを作り GET＠するルーチンです．そして，十字カーソルの時と同じ様に，このパターンを PUT で置いていきます．この処理を行っているのが＊INKSUB です．但し，図12の様に PUT で置く時には，〝OR〞を用います．これは，新しく描いたパターンの下にあるパターンに影響をおよぼさないためです．後は，この繰り返しにより，マジックで書くような効果が得られる訳です．また，このインキングを使う時には，十字カーソルの STEP 数を必ず 1 にしておかなければなりません．さもないと，点線になってしまいます．（＊MOVECUR の IF～THEN～ELSE）また，インキングに使うパターンは，画面上に線を描く時に太さを同じにするために，本当は円が望ましいのです（正方形であると斜めに移動した時に太さが太くなる）．しかし，今回の様に太さ10ぐらいですと，計算の誤差のために CIRCLE 命令を用いてもうまくいきません．ですから，今回は，正方形から角を削ってパターンを作りました．

リスト4

```
1000 '
1010 ' >>> Design (sample) [ (640 x 200) x 3 mode ] <<<
1020 '
1030 '
1040 ' ----->>>>> Disk version <<<<<-----
1050 '
1060 '
1070 '
```

```
1080 ON STOP GOSUB *ABORT
1090 STOP ON
1100 '
1110 ON ERROR GOTO *ERRTRAP
1120 '
1130 SCREEN 0,3:WIDTH 80,25:CONSOLE 0,17,0,1
1140 CLS 3:SCREEN 1,0,0,1 ←―― 白黒モード
1150 LOCATE 30,12:PRINT "--- Working ! ---"
1160 LOCATE 31,10:PRINT "*** Design ***"  円を描く時の角度のデータ
1170 DIM CU%(12),INCX(9),INCY(9),CIR(9),INK%(11) ←―― インキングの時に用いる
1180 DIM MTILE$(16),PIC%(1093),LSP%(4256) ←―――― 画面のセーブ・ロードに用いる
1190 XX=10:YY=10:X=110:Y=110:C=7:PSET1=0:INK=0:MC=0 } 変数のイニシャライズ
1200 X1=400:Y1=10:X2=423:Y2=44:INCX=20:INCY=10:C=7  } (変数領域を確保)
1210 GOSUB *CURSORINIT
1220 GOSUB *INKINIT
1230 GOSUB *CIRINIT
1240 GOSUB *TILEINIT
1250 GOSUB *XYLINE
1260 FOR I=1 TO 30:BEEP 1:BEEP 0:NEXT
1270 '                                                            メインルーチン
1280 *MAIN
1290   SCREEN 1,0,0,3:CONSOLE 0,17:GOSUB *MENU
1300   CLS:LOCATE,,0:CH=0:PSET1=0:INK=0:MC=0:C=7 ←― パレット番号
1310   SELECT=ASC(INPUT$(1))-47
1320   ON SELECT GOSUB *XYMOVE,*PSET1,*LINE1,*LINE2,*CIRCLE1,*DELETE1
*PAINT1,*PUTGET,*PRINTER,*LOADSAVE
1330 GOTO *MAIN
1340 '
1350 *MOVE
1360   X0=X:Y0=Y ←―― コマンドに入った時の座標の値を保存
1370   GOSUB *MOVECUR
1380 RETURN
1390 '
1400 *LINE1 (直線を描く)
1410   CLS:IF C=7 THEN COLOR@(50,20)-(56,20),3
1420   GOSUB *MOVE
1430   LINE (X0,Y0)-(X,Y),C
1440 RETURN
1450 '
1460 *LINE2 (四角を描く)
1470   CLS:IF C=7 THEN COLOR@(60,20)-(70,20),3
1480   GOSUB *MOVE
1490   LINE (X0,Y0)-(X,Y),C,B
1500 RETURN
1510 '
1520 *XYMOVE (Gカーソルの移動)
1530   CLS:COLOR@(50,19)-(56,19),3
1540   GOSUB *MOVECUR
1550 RETURN
1560 '
1570 *CIRINIT ←―― サークルのイニシャライズ
1580   RESTORE *CIRDATA
1590   FOR I=1 TO 9
1600     READ CIR(I) ←―― 角度のデータを読み込む
1610   NEXT
1620 RETURN
1630 '
1640 *CIRDATA
1650 DATA 0,0.785,1.57,2.355,3.14
1660 DATA 3.925,4.71,5.495,6.28
1670 '
1680 *CIRCLE1 (円を描く)
1690   CLS:COLOR@(50,21)-(57,21),3
1700   LOCATE,,1
1710   GOSUB *MOVE
1720   R=SQR((X-X0)^2+4*(Y-Y0)^2) ←―― 半径の計算
1730   IF R=0 THEN RETURN
1740   CLS:INPUT "start (1-9)";S
1750   CLS:INPUT "end   (1-9)";E
1760   GOSUB *XVIEW
1770   CIRCLE (SX,SY),R,C,CIR(S),CIR(E)
1780   C=7               ↖中心座標は，Gカーソル移動後の座標となる
1790 RETURN
```

初期画面

```
1800 '
1810 *PSET1（インキングの判断）
1820   CLS:LOCATE,,1:COLORQ(60,19)-(70,19),3
1830   INPUT "Set freehand [y/n] ";I$
1840   IF I$="y" THEN *INKING
1850   GOSUB *MOVECUR
1860   PSET (X,Y)
1870 RETURN
1880 '
1890 *INKING
1900   CLS:INPUT "size  (1,10) ";P$
1910   CLS:PRINT "End ------ Press (CR key) ------"
1920   IF P$="1" THEN PSET1=1:GOSUB *MOVECUR  ｝ペンの太さの選択
1930   IF P$="10" THEN INK=1:GOSUB *MOVECUR   ｝
1940 RETURN
1950 '
1960 *DELETE1 パレット番号を"0"に設定  (Delete)
1970   CLS:C=0:LOCATE,,1:COLORQ(60,21)-(70,21),3
1980   LOCATE 46,0:PRINT "       --- delete ---"
1990   LOCATE 46,2:PRINT "preset (1) or line (2)"                 ｝6つの消去の
2000   LOCATE 46,3:PRINT "    or lineb (3) or linebf (4)"         ｝仕方を選択
2010   LOCATE 46,4:INPUT "       or circle (5) or cls (6)";SEL    ｝
2020   CLS:ON SEL GOSUB *PRESET1,*LINE1,*LINE2,*DLINEBF,*DCIRCLE,*SCCL
S
2030 RETURN
2040 '
2050 *DLINEBF（画面の一部を消去）
2060   GOSUB *MOVE
2070   LINE (X0,Y0)-(X,Y),0,BF
2080 RETURN
2090 '
2100 *PRESET1（一点を消去）
2110   GOSUB *MOVECUR
2120   PRESET (X,Y)
2130 RETURN
2140 '
2150 *DCIRCLE
2160   C=0               ｝パレット番号を"0"にして円を消去
2170   GOSUB *CIRCLE1    ｝
2180 RETURN
2190 '
2200 *INKINIT（インキングのイニシャライズ）
2210   SCREEN 1,2:LINE (0,0)-(9,5),7,BF
2220   PRESET (0,0):PRESET (9,0)
2230   PRESET (0,5):PRESET (9,5)
2240   GET (0,0)-(9,5),INK% ←── ここで作られたパターンを読みこむ
2250   LINE (0,0)-(9,5),0,BF
2260 RETURN
2270 '
2280 *CURSORINIT（Gカーソルのイニシャライズ）
2290   SCREEN 1,2
2300   VIEW (0,0)-(55,25)    ＊Gカーソルについては
2310   LINE (0,10)-(50,10)   「Room」を参照
2320   LINE (25,0)-(25,20)
2330   GET (17,6)-(32,14),CU%
2340   CLS 2:SCREEN 1,0
2350 '
2360   RESTORE *CUDATA
2370   FOR I=0 TO 8
2380     READ INCX(I)
2390   NEXT
2400   FOR I=0 TO 8
2410     READ INCY(I)
2420   NEXT
2430 RETURN
2440 '
2450 *CUDATA
2460 DATA 1,10,20,30,40,50,60,70,80  ｝Gカーソルのステップ数のデータ
2470 DATA 1,5,10,15,20,25,30,35,40   ｝
2480 '
2490 *MOVECUR（Gカーソルの移動）
2500   LOCATE,,0:PRINT:MC=1
2510   PUT (X-8,Y-4),CU%,XOR
```

```
2520    GOSUB *GETKEY
2530  *SUBMCUR
2540    *LOOP1:IF CH=13 THEN *EXIT1      インキングの時には，Gカーソルのステップ数は，
2550      BEEP 1:BEEP 0:MC=1             常に"1"になる．他の場合は変えられる．
2560      IF CH=48 THEN INCX=INCX(0):INCY=INCY(0)
2570      IF PSET1 OR INK THEN INCX=1:INCY=1 ELSE IF (CH>=33) AND (CH=<
41) THEN DX=CH-32:INCX=INCX(DX):INCY=INCY(DX)
2580      LOCATE 10,24:PRINT USING "Cursor step  ##";INCX;
2590      COLOR@(23,24)-(25,24),6
2600      PUT (X-8,Y-4),CU%,XOR
2610      IF PSET1 THEN PSET (X,Y)
2620      IF INK=1 THEN GOSUB *INKSUB
2630      IF CH=49 THEN X=X-INCX:Y=Y+INCY  ┐
2640      IF CH=50 OR CH=31 THEN Y=Y+INCY  │
2650      IF CH=51 THEN X=X+INCX:Y=Y+INCY  │
2660      IF CH=52 OR CH=29 THEN X=X-INCX  │
2670      IF CH=53 THEN X=X:Y=Y            ├ Gカーソルの移動方向の判断
2680      IF CH=54 OR CH=28 THEN X=X+INCX  │
2690      IF CH=55 THEN X=X-INCX:Y=Y-INCY  │
2700      IF CH=56 OR CH=30 THEN Y=Y-INCY  │
2710      IF CH=57 THEN X=X+INCX:Y=Y-INCY  ┘
2720      IF X<10 THEN X=10
2730      IF X>370 THEN X=370     ┐
2740      IF Y<10 THEN Y=10       │
2750      IF Y>190 THEN Y=190     ├ Gカーソルの移動範囲を設定
2760      PUT (X-8,Y-4),CU%,XOR   ┘
2770      GOSUB *GETKEY
2780    GOTO *LOOP1
2790    *EXIT1:PUT (X-8,Y-4),CU%,XOR
2800    LOCATE,,1
2810  RETURN
2820  '
2830  *GETKEY（キー入力）
2840    CH=ASC(INPUT$(1))
2850  RETURN
2860  '
2870  *INKSUB（インキングの時の条件）
2880    IF X<14 THEN X=14
2890    IF Y<12 THEN Y=12
2900    IF X>365 THEN X=365
2910    IF Y>187 THEN Y=187
2920    PUT (X-4,Y-2),INK%,OR
2930  RETURN
2940  '
2950  *XYLINE（画面上に目盛りを振る）
2960    SCREEN 1,0,1,1 ←── 第2画面アクティブ
2970    FOR I=1 TO 19
2980      LINE (XX,10)-(XX,190),,,&HAAAA
2990      XX=XX+20
3000    NEXT
3010  '
3020    FOR I=1 TO 19
3030      LINE (10,YY)-(370,YY),,,&H8888
3040      YY=YY+10
3050    NEXT
3060  RETURN
3070  '
3080  *TILEINIT（ペイントのイニシャライズ）
3090    SCREEN 1,0,2,0 ←── 第3画面アクティブ
3100    RESTORE *TILEDATA
3110    FOR I=1 TO 16
3120      READ TL:TILE$=""
3130      FOR J=1 TO TL
3140        READ TL$:TILE$=TILE$+CHR$(VAL("&H"+TL$)) ←── タイル・ストリング作成
3150      NEXT
3160      VIEW (X1+1,Y1+1)-(X1+50,Y1+30),0,7
3170      PAINT (1,1),TILE$,7:MTILE$(I)=TILE$ ←── タイル・ストリング保存
3180      X1=X1+54
3190      IF X1>600 THEN X1=400:Y1=Y1+43
3200    NEXT
3210    GOSUB *KANINIT
3220  RETURN
3230  '
```

```
3240 *TILEDATA(タイル・パターンのデータ)
3250 DATA 1,ff
3260 DATA 2,aa,55
3270 DATA 4,88,00,22,00
3280 DATA 1,cc
3290 DATA 4,81,42,24,18
3300 DATA 8,01,03,07,0f,1f,3f,7f,ff
3310 DATA 8,11,22,44,88,88,44,22,11
3320 DATA 8,c0,60,30,18,0c,06,03,81
3330 DATA 8,03,06,0c,18,30,60,c0,81
3340 DATA 8,ff,ff,00,00,ff,ff,00,00
3350 DATA 8,f0,f0,f0,f0,0f,0f,0f,0f
3360 DATA 8,f7,e3,c1,80,00,80,c1,e3
3370 DATA 8,81,42,24,18,18,24,42,81
3380 DATA 8,ff,80,80,80,80,80,80,80
3390 DATA 8,00,18,24,42,42,24,18,00
3400 DATA 9,10,92,54,38,38,54,92,10,00
3410 '
3420 *KANINIT(漢字のイニシャライズ)  漢字ROMを使用し，タイル・パターンの名称を付ける
3430   SCREEN 1,0,2,0               注意{もし"漢字ROM"を使用していない方は，
3440   RESTORE *KANDATA                  このサブルーチンは必要ありません
3450   FOR I=1 TO 16                     このルーチンのかわりにリスト4-2のサブ
3460     READ K$                         ルーチンを使用してください}
3470     KCODE=VAL("&h"+K$)
3480     PUT (X2,Y2),KANJI(KCODE),PSET
3490     X2=X2+55
3500     IF X2>600 THEN X2=423:Y2=Y2+43
3510   NEXT
3520 '
3530 *KANDATA
3540 DATA 0161,0162,0163,0164,0165,0166
3550 DATA 0167,0168,0169,016a,016b,016c
3560 DATA 016d,016e,016f,0170
3570 RETURN
3580 '
3590 *PAINT1(ペイント)
3600   CONSOLE 0,25
3610   *LOOP2:IF CH=32 THEN *EXIT2
3620     CLS:SCREEN 1,0,0,5 ←── 第1，3画面を合成
3630     PUT (X-8,Y-4),CU%,XOR
3640     GOSUB *SUBMCUR
3650     LOCATE ,,0
3660     CLS:PRINT "Select tile pattern (a-p )"
3670     PRINT "     or  end (Press space key )"
3680     GOSUB *PGETKEY
3690     IF (TP<1) OR (TP>16) THEN RETURN
3700     GOSUB *XVIEW
3710     PAINT (SX,SY),MTILE$(TP),7
3720     GOSUB *GETKEY
3730   GOTO *LOOP2
3740 *EXIT2:RETURN
3750 '
3760 *PGETKEY(ペイント時のキー入力)
3770   TP=ASC(INPUT$(1))-96
3780 RETURN
3790 '
3800 *PRINTER(画面コピー)
3810   CLS:SCREEN 1,0,0,1:COLOR@(50,23)-(56,23),3
3820   PRINT "Prepare your printer."
3830   PRINT "     Press any key !"
3840   IF INKEY$="" THEN 3840
3850   CLS:COPY 2:"Copying now !!"
3860 RETURN
3870 '
3880 *PUTGET(GET&PUT)
3890   CLS:COLOR@(61,22)-(62,22),3
3900   COLOR@(63,22)-(65,22),2:LOCATE,,1
3910   PRINT "Load pattern [y/n] ?";
3920   L$=INPUT$(1)                        }外部記憶装置から，ロードするかを判断
3930   IF L$="y" THEN GOSUB *PLOAD
3940   CLS:PRINT "Already pattern [y/n] ?";
3950   A$=INPUT$(1)                        }現在，配列にあるパターンを使用するかを判断
3960   IF A$="y" THEN 4040
```

```
3970    CLS:GOSUB *MOVE      )
3980    LX=X:LY=Y            } GETする範囲を指定(Gカーソルによる)
3990    GOSUB *MOVE          )
4000    IF LX=X OR LY=Y THEN RETURN
4010    IF ABS(X-LX)>180 THEN RETURN
4020    IF ABS(Y-LY)>90 THEN RETURN
4030    GET (LX,LY)-(X,Y),PIC%:CH=0 ←── パターンをGETする
4040    COLOR@(63,22)-(65,22),7:COLOR@(69,22)-(71,22),2
4050    *LOOP3:IF CH=32 THEN *EXIT3
4060      SCREEN 1,0:LOCATE,,0
4070      PUT (X-8,Y-4),CU%,XOR ←── パターンをPUTする
4080      GOSUB *SUBMCUR
4090      GOSUB *XVIEW
4100      PUT (SX,SY),PIC%,OR
4110      LOCATE,,0
4120      GOSUB *GETKEY
4130    GOTO *LOOP3
4140    *EXIT3:LOCATE 0,0,1:CLS:PRINT "Do you save this pattern ? [y/n]
";
4150    S$=INPUT$(1) ← GETしたパターンを外部記憶装置にセーブするかを判断
4160    IF S$="y" THEN GOSUB *PSAVE
4170  RETURN
4180  '
4190  *PLOAD (パターンのロード)
4200    CLS:PRINT "+++ Load pattern +++"   (ディスクを使用していない方は,*PLOAD,*PSAVE
4210    GOSUB *MOVECUR:CLS:FILES:PRINT     の代わりに,リスト4.1の*CASLOAD,*CASSA
4220    INPUT "File name";N$               VEのルーチンを使用してください
4230    CLS:PRINT ">>> Reading <<<"        この時に,必ずラベル名は,統一して下さい)
4240    BLOAD N$,VARPTR(PIC%(0)):CLS
4250  RETURN 4040          配列に読み込む
4260  '
4270  *PSAVE (パターンのセーブ)                     ↙バイト数の計算
4280    CLS:INPUT"File name";N$:BYTE=(((X-LX+1)+7)¥8)*(Y-LY+1)*1+4
4290    CLS:PRINT ">>> Writing <<<"
4300    BSAVE N$,VARPTR(PIC%(0)),BYTE
4310  RETURN
4320  '                   配列のデータを書き込む
4330  *MENU
4340    LOCATE 50,18:PRINT "   *** select ***"
4350    LOCATE 50,19:PRINT "0.move      1.inking"
4360    LOCATE 50,20:PRINT "2.line      3.lineb"
4370    LOCATE 50,21:PRINT "4.circle    5.delete"
4380    LOCATE 50,22:PRINT "6.paint     7.get & put"
4390    LOCATE 50,23:PRINT "8.copy      9.load & save"
4400    LOCATE 50,24:PRINT "   END --- stop key  ";
4410    COLOR@(50,18)-(70,18),6:COLOR@(50,24)-(70,24),5
4420  RETURN
4430  '
4440  *ERRTRAP (エラー・トラップ)
4450    FOR I=1 TO 100:BEEP 1:BEEP 0:NEXT
4460    IF ERR=53 THEN CLS:PRINT "File not found !":RESUME *XWAIT
4470    IF ERR=56 THEN CLS:PRINT "Bad file name !":RESUME *XWAIT
4480    IF ERR=64 THEN CLS:PRINT "Disk I/O error !":RESUME *XWAIT
4490    IF ERR=68 THEN CLS:PRINT "Disk full !":RESUME *XWAIT
4500    IF ERR=69 THEN CLS:PRINT "Bad allocation table !":RESUME *XWAIT
4510    CLS:PRINT "ERROR !!!"
4520    FOR I=1 TO 1000:NEXT
4530    CLS:RESUME NEXT
4540
4550  *XWAIT
4560    FOR I=1 TO 1000:NEXT
4570  GOTO *MAIN
4580  '
4590  *SCCLS (画面のクリア)
4600    CLS:PRINT "--- Working ! ---"
4610    SCREEN 1,1,0,0
4620    VIEW (10,10)-(370,190),0
4630  RETURN
4640  '
4650  *LOADSAVE (画面のセーブとロード)
4660    CLS:LOCATE,,1:COLOR@(61,23)-(62,23),3
4670    SCREEN 1,0,0,1
4680    INPUT "Load (l) or save (s) or Cancel (c) ";SEL$
```

```
4690   IF SEL$='l' THEN GOSUB *XLOAD
4700   IF SEL$='s' THEN GOSUB *XSAVE
4710   IF SEL$='c' THEN RETURN
4720   IF SEL$<>'l' AND SEL$<>'s' AND SEL$<>'c' THEN *LOADSAVE
4730 RETURN
4740 '
4750 *XSAVE(画面のセーブ)
4760   COLOR@(69,23)-(73,23),2
4770   GET (10,10)-(370,190),LSP% ←── 作図領域をGETする
4780   CLS:INPUT 'File name ';N$            ┌ディスクを使用していない方は,
4790   CLS:PRINT '--- Writing ! ---'        │リスト4-1の*SCCASSAVE,
4800   BSAVE N$,VARPTR(LSP%(0)),&H2089      └*SCCASLOADを使用してください
4810 RETURN
4820 '
4830 *XLOAD(画面のロード)
4840   COLOR@(63,23)-(67,23),2:CLS:FILES:PRINT
4850   INPUT 'File name';N$
4860   CLS:PRINT '--- Reading ! ---'
4870   BLOAD N$,VARPTR(LSP%(0))
4880   CLS:PUT (10,10),LSP%,PSET ←── 読み込んだ図形をPUTする
4890 RETURN
4900 '
4910 *XVIEW (ビューポートを設定する)
4920   VIEW (10,10)-(370,190)←── 作図領域の限定
4930   SX=X-10:SY=Y-10 ←── PUT時などの補正
4940 RETURN
4950 '
4960 *ABORT
4970   IF MC THEN RETURN ←── Gカーソルが画面上に現れている時には,
4980   CONSOLE 0,25:CLS       STOPキーは受けつけない
4990   SCREEN 1,0,0,1
5000   LOCATE ,,1
5010   STOP OFF
5020 END
```

次に，ペイントについて説明しておきましょう．今回は，タイル・パターンを16個用意しました．これらは，いずれも4章で紹介したタイル･パターン･ジェネレータを使って拾いました（好みのパターンに変更した場合は，タイル・データの部分を変更してください）．これらのタイル・パターンの見本は，第3画面に描いておき，デザインしたパターンや文字にペイントする時に，第1画面と第3画面を合成して表示するようにしてあります．また，白黒モードでのタイリングは，1度行った後に，他のパターンのタイリングを行うことは，ほとんどの場合不可能なために，タイリングを行う前に作図したパターンや文字を保存しておく必要があります．この点には，充分気を付けてください．

同じパターンを何回も描くには大変手間がかかります．ですから，この様な場合は，作図したパターンをGET@で拾っておき，このパターンをPUT@で置いていき，同じ様なパターンを描きます．また，PUT@でパターンを置く場合，機能は好みによってなんでも良いのですが，今回は〝OR〞を取っています．もし，他の機能で使いたい場合は，好みによって書き換えてください．さらに，リスト4の*PUTGET を見て頂ければ分かると思いますが，ここで作ったパターンをディスク，あるいはテープにSAVE，またはLOADができるようにしてあります．このSAVEやLOADは高速に行える様に，4章のGエディタと同様，機械語プログラムとしてのロード／セーブを用いています．また，この時のスタート・アドレスは，〝VARPTR〞より得ます．詳しいことは，リスト4の*PLOAD，*PSAVE，あるいは4章の〝Gエディタ〞を参照してください．

さらに，これと同様に画面全体のSAVE／LOADもできます．これも，リスト4の*XSAVE，*XLOADを見て頂ければ理解できると思います．

以上に述べた機能が，今回用意したものです．これらの機能をうまく組み合わせることで，ある程度のパターンや文字はデザインできると思います．

6.4.2 Designの使い方

では最後に，今回のプログラムの使い方について説明しましょう．今回のプログラムでは，10種類のコマンドを用意しました．コマンドの入力の仕方は，0～9の数字で入力します．以下に各コマンドの機能と使用法について説明します．

○0 (move)

十字カーソルの移動を行います．操作の仕方は以下の通りです．

(方向) …………テン・キー（1～9，5を除く)，あるいはカーソル・キーを用いています．キーの対応は図13を見てください．

(STEP数) ……SHIFTキーを押しながら1～8の数字キーあるいは0の数字キーを用いています．キーと十字カーソルのSTEP数の対応も，図13を見てください．このコマンドから抜ける場合は，リターン・キーを押してください．

図13 カーソル移動キーとカーソル・ステップ・キー

○1 (inking)

PSETとフリー・ハンドがあります．フリー・ハンドを指定しないと，PSETモードになり1点のみの表示となります．また，フリー・ハンド・モードでは，ペンの太さが2種類（1と10)ありますので，どちらかを指定します．操作方法はMOVEに準拠します．この時，十字カーソルのステップ数を変えることはできません．常に1です．

START POINT (コマンド突入)
END POINT (コマンド終了)

図14 ライン の引き方

4 3 2
45°
5 1 (9)
6 7 8

○方向
START 3
END 5
START POINT
END POINT
(半径)

図15 円の角度指定

○2 (line)

直線を引くコマンドです.このコマンドに入った時の十字カーソルの位置が,直線を引く START POINT となります. 十字カーソルの操作法は, MOVEと同じで, END POINT の指定は, リターン・キーで行います (図14参照).

○3 (line box)

四角を描くコマンドです. 操作法は, line に準拠します.

○4 (circle)

円や半円を描くコマンドです. START POINT (コマンド突入時の十字カーソルの位置) と END POINT (リターン・キーを押した位置) の関係は以下の様になります.

i) (START POINT ー END POINT)=円の半径

ii) END POINT の位置が, 円の中心となる.

また, 円の描く範囲が方向で指定できます (45度ずつ). これは, 1~9の数字キーで行い, キーと方向の対応は図15の通りです.

〔例〕 円を描く場合は, 1, 9となります.

○5 (delete)

点, 線, 図形を消去するコマンドです.

このコマンドには, 6つのモードがありますので, 用途に応じて1~6の数字で選択します.

操作方法は, コマンド〝1〃から〝3〃までと全く同じです.

図16　GET／PUTの仕方

○ 6 (paint)

16種類のタイリングを行うコマンドです．

タイリングの方法は，十字カーソルを用いて塗りたい部分を指定（リターン・キーを押す）し，タイル・パターンをａ～ｐのアルファベットの小文字で指定します．また，タイリングを行った後に，スペース・キーを押すとコマンドが終了し，スペース・キー以外のキーを押すと，再びこのコマンドに突入します．

○ 7 (get & put)

このコマンドは，同一のパターンを何回も描くためのものです．まず，以前作ったパターンを LOAD（ディスクまたはカセットより）するかどうかを答えます．LOAD する場合は，十字カーソルにより LOAD 位置を指定し，パターンを LOAD します．LOAD した後にすぐ PUT モードに入ります．LOAD しない場合は，GET モードに入ります．十字カーソルにより，GET するパターンを図16の様に指定します．次に PUT モードに入ります．図16の様に，十字カーソルを用いて，PUT する位置を指定し描きます．PUT した後に，スペース・キー以外のキーを押すと再び PUT モードとなります．また，スペース・キーを押すと PUT モードから抜け SAVE モードに入ります．GET したパターンを SAVE するかどうかを答えてください．これを終えるとコマンドは終了です．GET モードか PUT モードかの判断は，画面上のメニューをみてください．どちらかの文字が赤くなります．また，GET する大きさに制限があり，最大で180×90ドットです．

○8 (copy)

画面のコピーを行います.

○9 (load & save)

画面の load あるいは save を行うコマンドです. load かsave かを答えてください. キャンセルする時は〝c〞を押してください.

○STOP キー

プログラムの実行を終了します. 但し, 十字カーソルが画面上にある時は受け付けません.

このプログラムは, 外部記憶装置としてディスクを想定しています. 外部記憶装置としてカセットを使用している方は, リスト 4 の＊PLOAD, ＊PSAVE, ＊XLOAD, ＊XSAVE の各ルーチンを DELETE し, リスト 4．1を最後に追加します. さらに, リスト 4の中で上記のルーチンを呼んでいるものすべてのラベル名をリスト 4．1のコメントに従って変更してください. このルーチンは, 内蔵の MONITOR を使用してテープの SAVE／LOAD を行うものです.

リスト4.1

```
6000 ' ************************************
6010 ' * Save & Load subroutine for cas.tape*
6020 ' ************************************
6030 '
6040 '
6050 '
6060 ' exchange:    *pload -----> *casload
6070 '              *psave -----> *cassave
6080 '              *xload -----> *sccasload
6090 '              *xsave -----> *sccassave
6100 '
6110 ' You must be ' GOSUB *CASINIT '
6120 '
6130 '
6140 '
6150 ' ----- INIT. for cas.tape -----
6160 '
6170 *CASINIT
6180   SADRS=0:EADRS=0:BYTE=0:X0=0:Y0=0:R=0:S=0:E=0
6190   SEL=0:I=0:J=0:TP=0:TL=0:LX=0:LY=0:SX=0:SY=0
6200   I$="":P$="":L$="":S$="":A$="":SEL$=""
6210 RETURN
6220 '
6230 ' ----- SAVE PATTERN for cas.tape -----
6240 '
6250 *CASSAVE
6260   BYTE=(((X-LX+1)+7)¥8)*(Y-LY+1)*1+4
6270   SADRS=VARPTR(PIC%(0))  ⎫アドレスの計算
6280   EADRS=SADRS+BYTE       ⎭
6290   CLS:PRINT ">>> Save pattern <<<"
6300   GOSUB *CASMON
6310 RETURN
6320 '
6330 ' ----- LOAD PATTERN for cas.tape -----
6340 '
6350 *CASLOAD
6360   CLS:PRINT ">>> Load pattern <<<"
6370   GOSUB *MOVECUR:CLS
```

```
6380   MON:CLS ←── モニタへ
6390 RETURN 4050
6400 '
6410 ' ---- SAVE SCREEN for cas.tape -----
6420 '
6430 *SCCASSAVE
6440   COLOR@(69,23)-(73,23),2
6450   GET (10,10)-(370,190),LSP%
6460   SADRS=VARPTR(LSP%(0))
6470   EADRS=SADRS+&H2089
6480   GOSUB *CASMON
6490 RETURN
6500 '
6510 ' ----- LOAD SCREEN for cas.tape -----
6520 '
6530 *SCCASLOAD
6540   COLOR@(63,23)-(67,23),2
6550   CLS:MON:CLS:PUT (10,10),LSP%,PSET
6560 RETURN ↖モニタへ
6570 '
6580 *CASMON
6590   LOCATE 48,0:PRINT "Start address ---> &H";HEX$(SADRS)
6600   LOCATE 48,1:PRINT "End   address ---> &H";HEX$(EADRS)
6610   MON:CLS
6620 RETURN
```

SAVE ……MONITORに入ったら，Wコマンドでセーブします．この時に，開始アドレス，終了アドレスは画面に表示された値を使用します．

LOAD ……MONITORに入ったら，Rコマンドでロードします．

以上のことが終了したら，CTRL-BでMONITORから抜けてください．この時点で，元のBASICプログラムに復帰します．

但し，カセットのルーチンを使用する前には，必ず1度だけ GOSUB ＊CASINIT を実行しておいてください．これを怠ると，図17のように変数の置かれているアドレスがずれてしまい，SAVE時とLOAD時における VARPTR で返ってくる値が一致しなくなりエラー，もしくは変数領域をメチャクチャに壊してしまいます．

図17　変数を初期設定しなければならない理由

また，漢字 ROM を使用していない方は，リスト4．2のルーチンをリスト4のコメントを参考にしながら挿入してください．この時に，＊KANINIT，＊KANDATA は，必要なくなりますから，DELETE してください．

リスト4.2

```
7000 '
7010 ' >>>>> If you have not "KANJI ROM"
7020 '              then you must use this subroutine <<<<<
7030 '
7040 *KANINIT
7050   LOCATE 53, 5:PRINT "a      b      c      d"
7060   LOCATE 53,11:PRINT "e      f      g      h"
7070   LOCATE 53,16:PRINT "i      j      k      l"
7080   LOCATE 53,22:PRINT "m      n      o      p"
7090 RETURN
```

6.5 ロジック回路を描く(Logic)

私達がロジック（論理）回路を紙面上で設計する時には，ロジック・テンプレートなどを用いて行いますが，実際に書いた経験のある人ならば，その労力は容易に推察できるでしょう．そこで，これもコンピュータにやらせてみます．

ロジック回路のコンピュータによる設計は，最近特に盛んになり，かなり進歩しています（このような応用は，CAD（Computer Aided Design）と呼ばれています）．専門分野では，入力装置（デジタイザ，ライト・ペンなど）や出力装置（CRT，XY プロッタなど）が充実しており，高度な手法が使われています．これをローコスト・コンピュータを用いて実現しようとすると，周辺装置に不利な点が多いため，専用機の様にはいきませんが，ソフトでカバーすればある程度のことは実現できると思います．

6.5.1 プログラムの説明

では，プログラムについてお話ししていきましょう．回路は，素子の形や直線が美しくないと，結果的に見にくいものになってしまいます．そこで今回は，PC-8801 の高分解能を利用して白黒400モードで使用することにしました．従って，高解像度の CRT をお持ちでない方は，残念ながら実行できません．

回路を画面上で描く原理は，Room や Design とほとんど変わりません．このプログラムも画面上の位置指定は，十字カーソルを用いて行います(十字カーソルについては6．1を参照してください)．また，この十字カーソルを，コマンドの選択にも使用しています（図18)．この様な十字カーソルによるコマンドの選択の方法は，ライト・ペンを意識したもので，コマンドの機能を図示することでコマンドの目的を明らかにし，選択を仕易くします．プログラムについては＊SEL-CUMOV を参照してください．また，コマンドの選択には，これに加えファンクション・キー(割り込みキー）も使用しています．このキーについては後述します．前節なども含めて，コマンド

図18 十字カーソルを用いたMENUの選択

図19 PUTする時の補正

図20 インバータのPUTの仕方
（切り絵の原理）

図21 第2のライン

の選択方法は色々ありますから，プログラムの用途に応じて使い易いものを選んでください．

素子（AND，OR，NOT）のパターンは，いずれもLINE文とCIRCLE文で描かれており，画面上に描く場合にはPUT@を用いています．但し，PUTする時にパターンと十字カーソルの関係が（図19）の様になっているために，補正する必要があります（＊ANDDRなど参照）．このことは，インバータや接続点も同様にPUT@を用いているため言えることです．但し，インバータを描く時には，描いた回路に影響がない様に，6．2の切り絵の原理を用いています．この時用意するパターンは，（図20）の様な2つのパターンです．詳しいことは，＊NOTLINEDR1を参照してください．

結線（直線）は，2種類用意しました．1つは単なる直線で，6．4のLINEと同じです．もう1つはクランク型の直線で，（図21）のような使い方をします．プログラムについては，＊LINEDRAW1を見て頂ければ理解できると思います．

また，十字カーソルの移動の仕方や，線や素子の消去の仕方は，6．4と同じです．

さらに，よく使用するパターン（例えばRSフリップフロップなど）が定義できるように，6．4と同様にGET@でパターンを拾い，配列の内容をBSAVEでSAVEすることができます．もちろんLOADも可能です．詳しいことは＊PTSAVE及び，＊PTLOADを参照してください．

図22　90°右回転

図23　右ローテーション方法

また，これと同じ方法で描いた全回路をSAVE／LOADすることもできます．但し，画面をGET／PUTする時に配列が足りなくなってしまうので，画面を4つに区切って画面をSAVE／LOADする様にしました（*SCSAVE，*SCLOAD）．

今回用意した素子は，すべて出力方向が右方向のものです．しかし回路を描く都合上，これだけではちょっと不便です．そこで，出力方向が上下左右の方向に向いたパターンも描けるようにしました．これには，それぞれの方向に向いたものを用意してもよいのですが，配列がかなり多く必要ですし，それぞれのパターンを作るのも大変です．

そこで，出力方向が右向きのものだけを用意し，他の方向が必要な時はこのパターンをローテーション（回転）させて描くようにしました．このパターンのローテーションは，4章で紹介した〝Glib〟を使用しています．このパッケージでは，X方向反転，Y方向反転，対称軸反転が用意されています．これらの3つを組み合わせることで，ローテーションが可能となります．ここでは右回転(90°)だけを用意しました(図22)．90°右回転をさせるには，機械語パッケージのPROTをコールし，対称反転を行った後に，PROTXをコールしX方向に反転させます(図23)．これでどのパターンに対しても，常に右回転します．ちなみに左回転は，PROTをコールし，次にPROTYをコールすることで実現されます．このプログラムでは右ローテーションは，*ELROT，*ELRIGHTの部分で行っています．詳しい内容については，リストのコメントを見てください．

また，ローテーションしたパターンを置く時に注意しなければならないことがあります．それは補正の問題です．〝Glib〟で用意した反転ルーチンでは，8bit（1バイト）単位の正方形であるため，ローテーションした時に中心がずれてしまいます．そこでPUTする時に各方向において補正しなければなりません．その補正を行っているのが，*ADJUSTとNOTLINEDR1のIF

文です．ここでは現在の出力方向を判断して，それなりの補正を行っています．そして，＊ROTINIT では，機械語パッケージを使用するためのイニシャライズを行っています．ここで重要なことは，使う変数の初期化です．これを実行しておかないと，VARPTR で不正確な値が返ってきてしまいます．その他の詳しい説明は，リスト5の横にあるコメントを見てください．

では最後にプログラムの使い方について説明しましょう．

リスト5

このプログラムを実行する前に必ず機械語のパッケージをロードすること

```
1000 '
1010 ' >>> logic  (sample)   [ (640 x 400) mode ] <<<
1020 '
1030 '
1040 ' ----->>>>> Disk version <<<<<-----
1050 '
1060 '
1070 '
1080 CLEAR ,&HE3FF:DEFINT A-Z:DEF USR=&HE400
1090 '
1100 ON ERROR GOTO *ERRTRAP
1110 '
1120 ON STOP GOSUB *ABORT
1130 STOP ON
1140 '
1150 ON HELP GOSUB *MESSAGE
1160 HELP ON
1170 '
1180 ON KEY GOSUB *ELROT,*ELROT1,*HCOPY,*PTLOAD,*PTSAVE,*SCLOAD,*SCSAV
E
1190 KEY ON
1200 '
1210 SCREEN 0,3:CLS 3:WIDTH 80,25:CONSOLE 0,19,0,1:COLOR 7,0,0,7
1220 PRINT "----- Working ! -----":COLOR 7
1230 DIM CU%(21),INCX(6),INCY(6),A%(257),O%(257)      } Gカーソル，素子，画面のセーブ，
1240 DIM N%(257),NT%(17),NT1%(17),P%(5),SCR%(3373)   } ロードに使用
1250 PAI!=3.14:CX=40:X=100:Y=100:SX=8:XK=430:INCX=10:INCY=10 } 変数のイニシャライズ
1260 XX=0:YY=0:DIR=0:DIR1=0:PARAX=10:PARAY=10              } (変数領域の確保)
1270 GOSUB *CURSORINIT
1280 GOSUB *MENU
1290 GOSUB *ROTINIT
1300 CLS:SCREEN 2,0:BEEP
1310 '                  ↖白黒400モード
1320 *SELECT                                              } メインルーチン
1330   LOCATE,,0:COLOR@(0,20)-(79,24),7:MC=0:XX=0:YY=0
1340   GOSUB *SELCUMOV :SEL=((CX-40)/70)+1 ←── コマンド選択
1350   ON SEL GOSUB *ANDDR,*ORDR,*NOTDR,*NOTLINEDR1,*LINEDRAW,*LINEDRA
W1,*MOVE,*DELETEL,*PSETER
1360 GOTO *SELECT
1370 '
1380 *ANDDR (AND素子を置く)
1390   COLOR@(1,20)-(8,24),3
1400   GOSUB *YVIEW:GOSUB *ADJUST
1410   PUT (X-31-XX,Y-31-YY),A%,PSET
1420   SCREEN 2,0   ↖補正をしている
1430 RETURN
1440 '
1450 *ORDR (ORを置く)
1460   COLOR@(10,20)-(17,24),3
1470   GOSUB *YVIEW:GOSUB *ADJUST
1480   PUT (X-31-XX,Y-31-YY),O%,PSET
1490   SCREEN 2,0
1500 RETURN
1510 '
1520 *NOTDR (NOTを置く)
1530   COLOR@(20,20)-(26,24),3
1540   GOSUB *YVIEW:GOSUB *ADJUST
1550   PUT (X-31-XX,Y-31-YY),N%,PSET
1560   SCREEN 2,0
1570 RETURN
1580 '
1590 *ADJUST
```

```
                                                                     3
                                                                     ↑
1600    IF (DIR MOD 4)=1 THEN XX=1 ｝ローテーションの時に
1610    IF (DIR MOD 4)=2 THEN YY=1 ｝中心がずれるために補正を行う  2←方向→0
1620  RETURN                                                         ↓
1630  '                                                              1
1640  *NOTLINEDR1（インバータを置く）
1650    COLOR@(29,22)-(32,23),3
1660    IF (DIR1 MOD 4)=0 THEN XX1=-1:YY1=8  ｝
1670    IF (DIR1 MOD 4)=1 THEN XX1=7:YY1=-1  ｝各方向において補正を行う
1680    IF (DIR1 MOD 4)=2 THEN XX1=16:YY1=7  ｝（方向の判断は，DIR1の値による）
1690    IF (DIR1 MOD 4)=3 THEN XX1=8:YY1=16  ｝
1700    PUT (X-XX1,Y-YY1),NT%,AND  ｝"cloth"の時の切り絵の原理を応用
1710    PUT (X-XX1,Y-YY1),NT1%,OR  ｝（周辺に影響を与えないために行う）
1720  RETURN
1730  '
1740  *PSETER（接続点を置く）
1750    COLOR@(74,22)-(76,23),3
1760    PUT (X-2,Y-2),P%,OR
1770  RETURN
1780  '
1790  *MOVE（Gカーソルの移動：MOVEコマンド）
1800    COLOR@(54,22)-(62,23),3
1810    GOSUB *MOVECUR
1820  RETURN
1830  '
1840  *LINEDRAW（結線を行う：普通のライン）
1850    COLOR@(36,22)-(44,23),3
1860    XX=X:YY=Y
1870    GOSUB *MOVECUR
1880    IF XX=X AND YY=Y THEN RETURN
1890    LINE (XX,YY)-(X,Y)
1900  RETURN
1910  '
1920  *LINEDRAW1（結線を行う：クランク型のライン）
1930    COLOR@(45,22)-(52,23),3
1940    XX1=X:YY1=Y
1950    GOSUB *MOVECUR
1960    IF XX1=X AND YY1=Y THEN RETURN
1970    POINT (XX1,YY1)         ｝
1980    LINE-((XX1+X)/2,YY1)    ｝
1990    LINE-((XX1+X)/2,Y)      ｝クランク型に結ぶ
2000    LINE-(X,Y)              ｝
2010  RETURN
2020  '
2030  *DELETEL（消去を行う）
2040    COLOR@(64,22)-(70,23),3:LOCATE,,1
2050    CLS:PRINT "--- delete ---"
2060    PRINT " 1. line"                      ｝
2070    PRINT " 2. clear (all)"               ｝消去の仕方の選択
2080    INPUT " 3. clear (part)    ";D        ｝
2090    CLS                                   ｝
2100    ON D GOSUB *DLINE,*ACLS,*PCLS
2110  RETURN
2120  '
2130  *DLINE（ラインの消去）
2140    XX=X:YY=Y
2150    GOSUB *MOVECUR
2160    LINE (XX,YY)-(X,Y),0
2170  RETURN
2180  '
2190  *ACLS（画面全体をクリア）
2200    LINE (0,0)-(639,333),0,BF
2210  RETURN
2220  '
2230  *PCLS（画面の一部をクリア）
2240    GOSUB *MOVECUR  ｝
2250    PX=X:PY=Y       ｝消去する範囲を指定
2260    GOSUB *MOVECUR  ｝
2270    IF PX=X OR PY=Y THEN RETURN
2280    LINE (PX,PY)-(X,Y),0,BF
2290  RETURN
2300  '
2310  *CURSORINIT（Gカーソルのイニシャライズ）
2320    SCREEN 2,2
```

```
2330   VIEW(0,0)-(55,25)         *Gカーソルについては
2340   LINE(0,10)-(50,10),7       "Room"を参照
2350   LINE(25,0)-(25,20),7
2360   GET (17,2)-(32,17),CU%
2370   CLS 2:SCREEN 2,3
2380 '
2390   RESTORE *CUDATA
2400   FOR I=1 TO 6
2410     READ INCX(I)
2420   NEXT
2430   FOR I=1 TO 6
2440     READ INCY(I)
2450   NEXT
2460 RETURN
2470 '
2480 *CUDATA
2490 DATA 1,10,20,30,40,50 } Gカーソルのステップ数のデータ
2500 DATA 1,10,20,30,40,50 } (400モード)
2510 '
2520 *MOVECUR (Gカーソルを移動する)
2530   LOCATE,,0:MC=1
2540   PUT (X-8,Y-8),CU%,XOR
2550   GOSUB *GETKEY
2560   *LOOP1:IF CH=13 THEN *EXIT1
2570     BEEP 1:BEEP 0
2580     IF (CH>47) AND (CH=<53) THEN DX=CH-47:INCX=INCX(DX):INCY=INCY
(DX)
2590     GOSUB *CSDISP
2600     PUT (X-8,Y-8),CU%,XOR
2610     IF CH=29 THEN X=X-INCX:IF X<8 THEN X=8          }
2620     IF CH=28 THEN X=X+INCX:IF X>631 THEN X=631      } Gカーソルの移動方向と
2630     IF CH=30 THEN Y=Y-INCY:IF Y<8 THEN Y=8          } 範囲を処理している
2640     IF CH=31 THEN Y=Y+INCY:IF Y>326 THEN Y=326      }
2650     PUT (X-8,Y-8),CU%,XOR
2660     GOSUB *GETKEY
2670   GOTO *LOOP1
2680   *EXIT1:PUT (X-8,Y-8),CU%,XOR
2690   LOCATE,,1:PARAX=INCX:PARAY=INCY ←── 現在のステップ数を保存
2700 RETURN
2710 '
2720 *GETKEY (キー入力)
2730   CH=ASC(INPUT$(1))
2740 RETURN
2750 '
2760 *SELCUMOV (Gカーソルによるコマンド選択)
2770   LOCATE,,0
2780   PUT (CX-8,358),CU%,XOR
2790   GOSUB *GETKEY
2800   *LOOP2:IF CH=13 THEN *EXIT2
2810     BEEP 1:BEEP 0:INCX=70:MC=0
2820     PUT (CX-8,358),CU%,XOR
2830     IF CH=29 THEN CX=CX-INCX:IF CX<40 THEN CX=600 }
2840     IF CH=28 THEN CX=CX+INCX:IF CX>600 THEN CX=40 } X軸方向に限定
2850     PUT (CX-8,358),CU%,XOR
2860     GOSUB *GETKEY
2870   GOTO *LOOP2
2880   *EXIT2:PUT (CX-8,358),CU%,XOR:INCX=PARAX:INCY=PARAY
2890 RETURN
2900 '
2910 *CSDISP
2920   LOCATE 60,19:PRINT USING "Cursor step  ##";INCX
2930   COLOR@(71,19)-(74,19),6
2940 RETURN
2950 '
2960 *MENU (コマンドの種類を画面上に表示し,またその描かれた素子のパターンを配列に読みこむ)
2970   POINT (20,356)
2980   GOSUB *ANDEL
2990   GET (9,335)-(72,398),A% ←── AND素子
3000   POINT (90,356)
3010   GOSUB *OREL
3020   GET (79,335)-(142,398),O% ←── OR素子
3030   POINT (170,366)
3040   GOSUB *NOTEL
```

```
3050    GET (154,335)-(217,398),N% ←── NOT素子
3060    POINT (240,366)
3070    GOSUB *NOTLINE1
3080    GET (240,358)-(255,373),NT1%   ┐
3090    PUT (576,270),NT1%,PSET        │
3100    PAINT (578,278),7,7            │インバータ
3110    GET (576,270)-(591,285),NT%    │2つのパターン(白ぬき，黒ぬきを作成)
3120    PUT (576,270),NT%,PRESET       │
3130    GET (576,270)-(591,285),NT%    │
3140    LINE (576,270)-(591,285),0,BF  ┘
3150    CIRCLE (600,369),2
3160    PAINT (600,369),7,7
3170    GET (598,367)-(602,371),P% ←── 接続点
3180    LINE (290,366)-(350,366) ┐
3190    POINT (360,356)          │
3200    LINE-STEP(25,0)          │2種類
3210    LINE-STEP(0,20)          │
3220    LINE-STEP(25,0)          ┘
3230    GOSUB *COMD
3240    LINE (0,334)-(639,399),,B
3250  RETURN
3260  '
3270  *ANDEL (ANDの作成 (ラインとサークルによる))
3280    LINE-STEP(0,-8)
3290    LINE-STEP(23,0)
3300    CIRCLE STEP(0,18),18,,3/2*PAI!,1/2*PAI!
3310    LINE STEP(0,18)-STEP(-23,0)
3320    LINE-STEP(0,-36)
3330    LINE-STEP(0,28)
3340    LINE-STEP(-6,0)
3350    LINE STEP(0,-20)-STEP(6,0)
3360    LINE STEP(43,10)-STEP(6,0)
3370  RETURN
3380  '
3390  *OREL (ORの作成)
3400    CIRCLE STEP(-24,10),25,,7/4*PAI!,1/4*PAI!
3410    LINE STEP(18,-18)-STEP(15,0)
3420    CIRCLE STEP(0,0),36,,3/2*PAI!,11/6*PAI!
3430    CIRCLE STEP(0,36),36,,1/6*PAI!,1/2*PAI!
3440    LINE-STEP(-15,0)
3450    POINT STEP(4,-8)
3460    LINE-STEP(-6,0)
3470    LINE STEP(0,-20)-STEP(6,0)
3480    LINE STEP(43,10)-STEP(6,0)
3490  RETURN
3500  '
3510  *NOTEL (NOTの作成)
3520    LINE STEP(31,0)-STEP(-31,-18)
3530    LINE-STEP(0,36)
3540    LINE-STEP(31,-18)
3550    LINE-STEP(6,0)
3560    LINE STEP(-39,0)-STEP(-6,0)
3570  RETURN
3580  '
3590  *NOTLINE1 (インバータの作成)
3600    CIRCLE STEP(3,0),3
3610    LINE STEP(3,0)-STEP(9,0)
3620  RETURN
3630  '
3640 ┌*COMD (コマンド(MOVE, DELETE)の表示)
3650 │  RESTORE *COMDDATA
3660 │  FOR I=1 TO 8          ┌"漢字ROM"を使用していない方は，
3670 │    READ A$             └このルーチンの代わりに，リスト5.2を使用してください
3680 │    KCODE=VAL("&h"+A$)
3690 │    PUT (XK,360),KANJI(KCODE),PSET
3700 │    XK=XK+16
3710 │  NEXT
3720 │RETURN
3730 │'
3740 │*COMDDATA
3750 │DATA 234d,234f,2356,2345,2340
3760 │DATA 2344,2345,234c
3770 └'
```

```
3780 *SCSAVE（画面のセーブ）            ［*テープを使用している方は，①の部分の代わりに
3790   IF MC THEN GOTO *INHIBIT             GOSUB  *SCCASSAVEとしてください］
3800   KEY OFF:LOCATE,,1
3810   CLS:PRINT "*** save screen ***"
3820  ┌LOCATE 0,2:INPUT "File name [2:] ";N$
3830  │CLS:PRINT "----- Writing !! -----"
3840  │FOR I=1 TO 4
3850 ①│  GET (SX,0)-(SX+155,336),SCR%                       ｝画面全体をGETできない
3860  │  BSAVE "2:"+N$+STR$(I),VARPTR(SCR%(0)),&H23F8       ｝ために画面を4つに分けて
3870  │  SX=SX+156 ＼ドライブ2を使用                          ｝セーブしている
3880  └NEXT I
3890   SX=8:CH=0:LOCATE,,0
3900   KEY ON:CLS:BEEP
3910 RETURN
3920 '                                  ［*テープを使用している方は，
3930 *SCLOAD（画面のロード）              ②の部分の代わりに，GOSUB  *SCCASLOAD
3940   IF MC THEN GOTO *INHIBIT          としてください］
3950   KEY OFF:LOCATE,,1
3960   CLS:PRINT "*** load screen ***"
3970  ┌LOCATE 0,2:FILES 2:PRINT:INPUT "File name [2:] ";N$
3980  │CLS:PRINT "----- Reading !! -----"
3990  │FOR I=1 TO 4 ←── 画面を4つに分けロード．
4000 ②│  BLOAD "2:"+N$+STR$(I),VARPTR(SCR%(0))
4010  │  PUT (SX,0),SCR%,PSET
4020  │  SX=SX+156
4030  └NEXT I
4040   SX=8:CH=0:LOCATE,,0
4050   KEY ON:CLS:BEEP
4060 RETURN
4070 '
4080 *MESSAGE（HELPキーのためのメッセージ）
4090   HELP OFF:COLOR 4
4100   CLS:PRINT " (f-1 key & CR)-----> Rotation [A,O,N]"
4110   PRINT " (f-2 key & CR)-----> Rotation [Inv.]"
4120   PRINT " (f-3 key & CR)-----> Copy"
4130   PRINT " (f-4 key & CR)-----> Load pattern"
4140   PRINT " (f-5 key & CR)-----> Save pattern"
4150   PRINT " (f-6 key & CR)-----> Load screen"
4160   PRINT " (f-7 key & CR)-----> Save screen"
4170   PRINT SPACE$(13);" ***  (STOP KEY)----------> END  ***"
4180   PUT (X-8,Y-8),CU%,XOR
4190   FOR I=0 TO 5000:NEXT
4200   PUT (X-8,Y-8),CU%,XOR
4210   COLOR 7:CLS
4220   HELP ON
4230 RETURN
4240 '
4250 *YVIEW（ビューポートを設定する）
4260   VIEW (0,0)-(639,333)
4270 RETURN
4280 '
4290 *ERRTRAP（エラー・トラップ：Disk関係について強化してあります）
4300   COLOR 2:FOR I=1 TO 100:BEEP 1:BEEP 0:NEXT
4310   IF ERR=53 THEN CLS:PRINT "File not found !":RESUME *XWAIT
4320   IF ERR=56 THEN CLS:PRINT "Bad file name !":RESUME *XWAIT
4330   IF ERR=61 THEN CLS:PRINT "File write protected !":RESUME *XWAIT
4340   IF ERR=62 THEN CLS:PRINT "Disk offline !":RESUME *XWAIT
4350   IF ERR=64 THEN CLS:PRINT "Disk I/O error !":RESUME *XWAIT
4360   IF ERR=68 THEN CLS:PRINT "Disk full !":RESUME *XWAIT
4370   IF ERR=69 THEN CLS:PRINT "Bad allocation table !":RESUME *XWAIT
4380   IF ERR=72 THEN CLS:PRINT "Delete record !":RESUME *XWAIT
4390   CLS:PRINT "Beyond the limits ."
4400   PRINT "    Move cursor !!"
4410   FOR I=0 TO 3000:NEXT:COLOR 7:CLS
4420   RESUME NEXT
4430 '
4440 *XWAIT
4450   FOR I=0 TO 1000:NEXT
4460   KEY ON:COLOR 7:CLS
4470   GOSUB *XORPUT
4480 GOTO *SELECT
4490 '
4500 *XORPUT
```

```
4510     PUT (CX-8,358),CU%,XOR
4520   RETURN
4530   '
4540   *INHIBIT (STOPキーを禁止します)
4550     FOR I=1 TO 30:BEEP 1:BEEP 0:NEXT
4560   RETURN
4570   '                              ※テープを使用している人は,
4580   *PTSAVE (パターンのセーブ)       ③の部分の代わりに,GOSUB *CASSAVEとしてください
4590     IF MC THEN GOTO *INHIBIT      ④の部分の代わりに,GOSUB *CASLOADとしてください
4600     KEY OFF:INCX=PARAX:INCY=PARAY
4610     CLS:PRINT ">>> Save Pattern <<<"
4620     GOSUB *MOVECUR
4630     XP0=X:YP0=Y                  セーブする範囲を指定する
4640     GOSUB *MOVECUR
4650     IF X=XP0 OR Y=YP0 THEN GOTO 4730
4660     XD=ABS(X-XP0):IF XD>300 THEN GOTO 4730
4670     YD=ABS(Y-YP0):IF YD>170 THEN GOTO 4730
4680     BYTE=(((XD+1)+7)¥8)*(YD+1)*1+4 ←── バイト数の計算
4690     LOCATE 0,2:INPUT "File name [2:]";N$
4700     GET (XP0,YP0)-(X,Y),SCR% ←── パターンのGET
4710 ③  CLS:PRINT "----- Writing !! -----"
4720     BSAVE "2:"+N$,VARPTR(SCR%(0)),BYTE ──ドライブ2を使用
4730     SX=8:CH=0:LOCATE,,0
4740     KEY ON:CLS:BEEP
4750   RETURN
4760   '
4770   *PTLOAD (パターンのロード)
4780     IF MC THEN GOTO *INHIBIT
4790     KEY OFF:INCX=PARAX:INCY=PARAY
4800     CLS:PRINT ">>> Load Pattern <<<"
4810     GOSUB *MOVECUR ←── PUTする場所を指定する
4820     LOCATE 0,2:FILES 2:PRINT:INPUT "File name [2:]";N$
4830 ④  CLS:PRINT "----- Reading !! -----"
4840     BLOAD "2:"+N$,VARPTR(SCR%(0))
4850     PUT (X,Y),SCR%,OR ←── パターンのPUT
4860     SX=8:CH=0:LOCATE,,0
4870     KEY ON:CLS:BEEP
4880   RETURN
4890   '
4900   *HCOPY (画面をコピーする)
4910     IF MC THEN GOTO *INHIBIT
4920     COLOR 5:CLS:PRINT "Prepare your printer ."
4930     PRINT "    Press any key  !"
4940     IF INKEY$="" THEN 4940
4950     CLS:PRINT "Copying now !!":COPY 4:COLOR 7:CLS
4960   RETURN
4970   '
4980   *ROTINIT (ローテーションのためのイニシャライズ)
4990     PARAM=&HE5F8:PROT=11:PROTX=12:PROTY=13          POKEを用いて
5000     W=0:PW=VARPTR(W):PP1=0:PP2=0:DUMY=0             直接,書き込んでいる
5010     W=VARPTR(PP1):POKE PARAM  ,PEEK(PW):POKE PARAM+1,PEEK(PW+1)
5020     W=VARPTR(PP2):POKE PARAM+2,PEEK(PW):POKE PARAM+3,PEEK(PW+1)
5030   RETURN
5040   '
5050   *DIMENSION (ローテーションする範囲の大きさを定義している)
5060     LOCATE 0,0:PRINT "----- Working !!!!! -----"
5070     MAX=NBYTE*NBYTE*4+1:DIM P1(MAX),P2(MAX)
5080   RETURN
5090   '
5100   *ELROT(AND,OR,NOTを右回りにローテーションする)
5110     GOSUB *XORPUT:CX=460:GOSUB *XORPUT ←── 強制的に"MOVE"へ
5120     NBYTE=8:RX=639:RY=333:GOSUB *DIMENSION
5130     PP1=VARPTR(P1(0)):PP2=VARPTR(P2(0))
5140     PUT (576,270),A%,PSET ←── ワーキングエリアに,ANDを置く
5150     GOSUB *RIGHTROT ←── ローテーションする
5160     PUT (10,335),P2,PSET ←── コマンド表示を変える
5170     FOR I=0 TO 257:A%(I)=P2(I):NEXT ←── ローテーションされたパターンを配列に読み込む
5180   '
5190     PUT (576,270),O%,PSET ←── OR
5200     GOSUB *RIGHTROT
5210     PUT (80,335),P2,PSET
5220     FOR I=0 TO 257:O%(I)=P2(I):NEXT
5230   '
```

```
5240    PUT (576,270),N%,PSET ←── NOT
5250    GOSUB *RIGHTROT
5260    PUT (150,335),P2,PSET
5270    FOR I=0 TO 257:N%(I)=P2(I):NEXT
5280    DIR=DIR+1:LINE (576,270)-(639,333),0,BF ←── 現在の出力方向を記憶しておき
5290    ERASE P1,P2:CH=0:CLS                         ワーキングエリアを消去する
5300 RETURN
5310 '
5320 *ELROT1 (インバータをローテーションする(*ELROT参照))
5330    GOSUB *XORPUT:CX=460:GOSUB *XORPUT
5340    NBYTE=2:RX=591:RY=285:GOSUB *DIMENSION
5350    PP1=VARPTR(P1(0)):PP2=VARPTR(P2(0))
5360    PUT (576,270),NT1%,PSET
5370    GOSUB *RIGHTROT
5380    PUT(240,358),P2,PSET
5390    FOR I=0 TO 17:NT1%(I)=P2(I):NEXT
5400    PUT (576,270),NT%,PSET
5410    GOSUB *RIGHTROT
5420    FOR I=0 TO 17:NT%(I)=P2(I):NEXT
5430    DIR1=DIR1+1:LINE (576,270)-(591,285),0,BF
5440    ERASE P1,P2:CH=0:CLS
5450 RETURN
5460 '
5470 *RIGHTROT (右回りのローテーション)
5480    GET (576,270)-(RX,RY),P1 ── 対称軸反転(機械語パッケージ)をコール
5490    DUMY=USR(PROT):PUT(576,270),P2,PSET
5500    GET (576,270)-(RX,RY),P1:DUMY=USR(PROTX) ←── X軸反転をコール
5510 RETURN
5520 '
5530 *ABORT
5540    IF MC THEN RETURN ←── Gカーソルが画面上にある場合は,STOPキーを禁止
5550    LOCATE,,1:CONSOLE 0,25
5560    GOSUB *XORPUT
5570    CLS:SCREEN 2,0
5580    STOP OFF
5590    KEY OFF
5600 END
```

Cursor step 20

MOVE DEL

Cursor step 1

6.5.2 Logicの使い方

今回のプログラムでは，コマンドの選択には，十字カーソルとファンクション・キーを用います．回路を描く時には十字カーソルを，ローテーションや画面のSAVE／LOADには，ファンク

ション・キーを使用しています.

まず，十字カーソルによるコマンドの選択の仕方についてお話ししましょう．十字カーソルの操作法はカーソル・キーを用いて行い，カーソルの移動方向はx軸（水平）方向のみです．コマンドを選択するには，十字カーソルを目的のコマンドの図まで移動させ，リターン・キーを押すことにより可能です．まず，十字カーソルによるコマンドの機能について説明しましょう．

○素子（AND，OR，NOT）

素子を画面上に描くコマンドです．現在の十字カーソルの位置に素子を置きます．また，f･1キーにより出力方向を変えることができます．

○インバータ

インバータ（丸印）を描くコマンドです．現在の十字カーソルの位置に置きます．インバータに付属しているラインは，中心の目安としてつけたもので，結線する時に大いに役立ちます．また，f･2 キーにより付属ラインの方向が変わります．

○ライン

結線をするコマンドです．結線する操作法は，6．4のLineに準拠します．結線する時に，素子やインバータに付属しているラインを目安に使うと美しく引くことができます．

○ライン（クランク型）

もう1つのラインと形が違う他は，全く同じです．

○MOVE

十字カーソルを移動させるコマンドです．十字カーソルの操作は，カーソル・キーを用いて行い，十字カーソルのSTEP数はテン・キー（0～5）を用いて変えます（図24)．他の事柄については，6．4のMOVEに準拠します．

十字カーソルの移動
↑
← → ↓
コマンドの選択
4 5
1 2 3
0
×10
STEP数

図24　十字カーソルの操作

○接続点

接続点（白丸印）を画面上の現在の十字カーソルの位置に描きます．

では次に，ファンクション・キーのコマンドの機能について説明しましょう．ファンクション・キーは，f･1からf･7まで使用しており，ファンクション・キーを押した後に，リターン・キーなどのなんらかのキーを押さないと，コマンドを実行しません(割り込みを使用している)．但し，ファンクション・キーは，カーソルがコマンド・モードにある時のみ有効です．

○ (f･1)

素子のローテーションを行います．ローテーションは，1度に90°ずつ右回転です．

○ (f･2)

インバータのローテーションを行います．やはりこれも90°右回転です．

○ (f･3)

画面のコピーを行います．

○ (f･4)

以前に作ったパターンをロードします．位置指定は十字カーソルにより行います．

○ (f･5)

RS･フリップフロップなどの小さなパターンをセーブします．セーブの仕方は，セーブする領域を十字カーソルを用いて指定して行います．また，セーブ領域は最大で300×170ドットです．

○ (f･6)

以前に描いた回路図をロードします．

○ (f･7)

現在の画面上の回路図をセーブします．

○STOP キー

プログラムの実行を終了します．

○HELP キー

ファンクション・キーの説明と現在の十字カーソルの位置が表示されます．

SAVE／LOADは，すべてドライブ2を使用します（ディスクを使用している場合）から，ド

ライブ2にフォーマットしたディスクを入れておいてください．

注意：今回のリスト5はディスク用です．

テープを使用している方は，リスト5の*SCSAVEの1部をGOSUB　*SCCASSAVEに，*SCLOADの1部をGOSUB　*SCCASLOADに，*PTSAVEの1部をGOSUB　*CASSAVEに，*PTLOADの1部をGOSUB　*CASLOADに，それぞれ変更し，リスト5．1をリスト5の最後に追加してください．さらに，カセットのルーチンを使う場合は，"Design"で述べた様な理由により，呼び出す前に必ず1度だけGOSUB　*CASINITを実行しておいてください．

また，漢字ROMを使用していない方は，リスト5の*COMD，*COMDDATAをDELETEし，リスト5の最後にリスト5．2を追加します．

上記の変更は，リスト中のコメントを参考に行ってください．

このプログラムでは，原則として400モード専用にプログラムを組んでありますが，もしどうしても200モードで使用する場合には，パターン（素子や十字カーソルなど）のy方向の大きさを½にして使用してください．但しこの場合には，ローテーションの機能は使用できなくなるので，取り除いてください．

（1）素子を置く

f-2を2回使用

（2）インバータを置く

（2）′結線する

（3）結線する

（3）′インバータを置く

（4）パターンのセーブ〔f-5〕

inverter

MOVEコマンド
で位置を決める

接続点コマンド
を選ぶ

（補足）

図25　回路図をセーブする方法

では最後に，回路図の描き方について説明しましょう．図25を見てください．まず MOVE コマンドを使用して十字カーソルを望む位置に移動し，素子を置きます．次に，十字カーソルを調整し，インバータを置き，結線して完成です．この時，結線をしてからインバータを置いても構いません．最後に，よく使用するパターンや保存しておきたい回路図をセーブしておくとよいでしょう．

今回は，コマンド数が多いためにHELPキーを使用してファンクション・キーの内容の説明を一定時間するようにしました．6.4でも説明しましたが，このようなプログラムでは，必ずエラー・トラップを行って，実行が中止しないように心がけてください．

今回のプログラムでは，多入力の素子や抵抗，コンデンサなどが用意されていないため，あまり実用的とは言えませんし，スケール（素子の大きさを変化させる）や文字入力などの機能も不足しています．このような点を考慮して皆さんも自分なりの強力なプログラムを考えてみてはいかがでしょうか．

リスト5.1

テープを使用している方は，リスト5の最後に，これらの5つのサブルーチンを，挿入してください

```
6000 ' ***********************************
6010 ' * Save & Load subroutine for cas.tape*
6020 ' ***********************************
6030 '
6040 '  exchange :    *ptload -----> *casload
6050 '   (part)       *ptsave -----> *cassave
6060 '                *scload -----> *sccasload
6070 '                *scsave -----> *sccassave
6080 '
6090 '  You must be ' GOSUB *CASINIT '
6100 '
6110 '  ----- INIT. for cas.tape -----
6120 '
6130 *CASINIT
6140   SADRS=0:EADRS=0:BYTE=0:DIR=0:DIR1=0:XX1=0:YY1=0
6150   D=0:PX=0:PY=0:XP0=0:YP0=0:NBYTE=0:RX=0:RY=0:I=0
6160   SEL=0:DX=0:XD=0:YD=0
6170 RETURN
6180 '
6190 '  ----- SAVE PATTERN for cas.tape -----
6200 '
6210 *CASSAVE (パターンのセーブ)
6220   GET(XP0,YP0)-(X,Y),SCR%
6230   SADRS=VARPTR(SCR%(0))
6240   EADRS=SADRS+BYTE
6250   GOSUB *CASMON
6260 RETURN
6270 '
6280 '  ----- LOAD PATTERN for cas.tape -----
6290 '
6300 *CASLOAD (パターンのロード)
6310   MON:CLS
6320 RETURN
6330 '
6340 '  ---- SAVE SCREEN for cas.tape -----
6350 '
6360 *SCCASSAVE (画面のセーブ)
6370   FOR I=1 TO 4
6380     GET (SX,0)-(SX+155,336),SCR%
6390     SADRS=VARPTR(SCR%(0))
6400     EADRS=SADRS+&H23F8
6410     GOSUB *CASMON ←── 多少，面倒ですが
6420     SX=SX+156          4回モニタへ入ります
6430   NEXT
6440 RETURN
6450 '
6460 '  ----- LOAD SCREEN for cas.tape -----
```

```
6470                                    
6480 *SCCASLOAD'(画面のロード)
6490   FOR I=1 TO 4
6500     MON:CLS ←
6510     PUT(SX,0),SCR%,PSET
6520     SX=SX+156
6530   NEXT
6540 RETURN
6550 '
6560 '
6570 '
6580 *CASMON
6590   PRINT "Start address ---> &H";HEX$(SADRS) } アドレスの表示
6600   PRINT "End   address ---> &H";HEX$(EADRS) }
6610   MON:CLS ←── モニタへ
6620 RETURN
```

リスト5.2

```
7000 '
7010 ' >>>>> If you have not "KANJI ROM"
7020 '                then you must use this subroutine <<<<<
7030 '
7040 *COMD
7050   LOCATE 55,23:PRINT "MOVE      DELETE"
7060 RETURN
```

6.6 ポリゴンの利用(Sango)

ポリゴン（多角形）を利用しての作図は，本格的グラフィックスの世界でもよく使われています．今回は，ポリゴンといっても，リスト6で用いているのは図26に示してあるような，直角二等辺三角形と正方形を組み合わせた五角形です．これをあるルールに従って，ある数値を参考につなぎ合わせていくと，木やサンゴが，伸びていくような模様を描くことができます．

図26　1つの五角形

図27　五角形による枝の伸び方

図28　枝の伸びる方向とその値

あるルールの1つは，伸びる方向が必ず決まっていることです．伸びる時は，必ず図26に示した五角形の辺 AB に，次の五角形の辺 D′C′ がつながっていきます（図27参照）．どこまで伸びていくかの上限を設定する変数が UPPER で，プログラムを RUN させた時，

BRANCH LENGTH :

と聞かれた時に答えた値が採用されます．極端に大きな値ですと，プログラムの実行が遅いため，なかなか図形が完成しませんから，1桁程度の値を代入してください．

次のルールは，枝分れのルールです．変数 UPPER で示された長さまで枝を伸ばしきってしまうと，その最後に描いた五角形の辺 AE に当たる部分から，今まで描いた枝と全く同じ形で縮小された枝が，前と同様に変数 UPPER で示された長さ分だけ伸びていきます．これを繰り返していると，無限に枝分れをしていきプログラムの実行が終りません．そこで，何回枝分れをするかをコントロールする変数が LIMIT です．これも，

BRANCH LEVEL :

と聞かれた時に答えた数値が，変数 LIMIT に1を引いてから代入されますが，変数 UPPER と同様に，あまり大きな値を入力しない方が無難です．1つの五角形は，必ずD→E→A→B→C→Dの順に描くので，点Dと伸びる方向が分かれば，枝を描くことができます．その点Dのx，y座標を変数X，Y，枝の伸びる方向（図28参照）を変数 VECTOR，五角形の辺 DE の長さを変数Eに代入して，＊PENTAGON を呼び出すと，スタックを指し示すスタック・ポインタ（変数 SP）の値で決定する色で五角形を描きます．

これらの動作は，スタックと呼ばれる一時的な記憶場所を用いることによって制御されます．各五角形を描く度に，スタックの代わりをする配列 STACK に，次の様な変数を積み重ねていきます．

P : **枝分れをする時のx座標．**
Q : **枝分れをする時のy座標．**
V : **枝分れをした後，枝の伸びる方向．**
E : **その五角形の大きさ．**
LEVEL : **枝分れの度合い（幹から何回枝分れをしたか）を示す．**

スタックにこれらの値を積み重ねるには，サブルーチン＊PUSH を呼び出します．スタックに積んだ値を用いて，五角形を描くには，サブルーチン＊PULL を呼び出すと，＊PENTAGON で用いる変数が，セットされます．変数 LEVEL は，各枝が伸びている時々で，スタックに記憶してあるため，現在，どのくらい幹（一番最初に描いた枝）から枝分れしたかが分かるので，その値と変数 LIMIT の持っている値と比較して，次の枝分れをするかしないかを決定します．このように，スタックを用いると再帰的なプログラム（この場合，同じ形の枝が，節のそれぞれから

枝分れし，その枝のそれぞれの節から枝分れし，……といったプログラム)が，BASICでも実現できます．

リスト6

```
1000 '
1010 '  >>> sango <<<
1020 '
1030  GOSUB *INITIALIZE
1040  GOSUB *MAIN
1050 '
1060 END
1070 '
1080 *INITIALIZE
1090  DIM MATRIX(7,1,3),PM(7,1),STACK(1000)
1100 '
1110  X=0 : Y=0 : P=0 : Q=0 : E=100
1120  SP=0 : VECTOR=0 : LEVEL=0
1130 '
1140  SCREEN ,3 : CLS
1150  INPUT "BRANCH LENGTH : ",UPPER
1160  UPPER=UPPER-1
1170  INPUT "BRANCH LEVEL   : ",LIMIT
1180  LIMIT=LIMIT-1
1190 '
1200  RESTORE *ARRAY
1210  FOR F=0 TO 7
1220    FOR G=0 TO 3
1230      READ MATRIX(F,0,G),MATRIX(F,1,G)
1240    NEXT
1250  NEXT
1260 '
1270  FOR F=1 TO 7 STEP 2
1280    FOR H=0 TO 3
1290      FOR G=0 TO 1
1300        MATRIX(F,G,H)=SGN(MATRIX(F,G,H))/SQR(2)/2
1310      NEXT
1320    NEXT
1330  NEXT
1340 '
1350  RESTORE *ARRAY1
1360  FOR F=0 TO 7
1370    READ PM(F,0),PM(F,1)
1380  NEXT
1390 '
1400  FOR F=1 TO 7 STEP 2
1410    FOR H=0 TO 1
1420      PM(F,H)=SGN(PM(F,H))/SQR(2)/2
1430    NEXT
1440  NEXT
1450 '
1460  SCREEN 0,1 : CLS 3
1470  WINDOW(-249,-449)-(400,20)
1480  VIEW(40,1)-(635,198),0,7
1490  LOCATE 6,0 : PRINT "Loop  Level  Stack"
1500 '
1510  ON ERROR GOTO *TRAP
1520 RETURN
1530 '
1540 *MAIN
1550  FOR I=0 TO UPPER
1560    GOSUB *PENTAGON
1570 '
1580    P=X+MATRIX(VECTOR,0,0)*E
1590    Q=Y-MATRIX(VECTOR,1,0)*E
1600    X=X+MATRIX(VECTOR,0,0)*E+MATRIX(VECTOR,0,1)*E
1610    Y=Y-MATRIX(VECTOR,1,0)*E-MATRIX(VECTOR,1,1)*E
1620 '
```

(1210〜1250) 8つの方向それぞれの五角形の塗り始めを決定するデータの読み込み

(1280〜1320) 上のFOR～NEXTループで読み込んだデータの補正

(1350〜1380) 8つの方向それぞれの五角形の塗り始めを決定するデータの読み込み

(1400〜1440) 上のFOR～NEXTループで読み込んだデータの補正

(1580〜1610) 主要座標の計算

```
1630     VECTOR=(VECTOR+1) MOD 8 ←── 五角形の延びる方向を,45°右に回す
1640    'V=(VECTOR+6) MOD 8
1650
1660     IF (VECTOR AND 1)=1 THEN E=E*SQR(2) ELSE E=E*SQR(2)/4
1670     GOSUB *PUSH
1680     LOCATE 6,1 : PRINT USING " ##    ###    ###";I,LEVEL,SP/5
1690   NEXT
1700  '
1710   GOSUB *PULL
1720   IF LEVEL<LIMIT THEN LEVEL=LEVEL+1 : GOTO *MAIN
1730   REG=LEVEL
1740   FOR I=0 TO UPPER
1750     GOSUB *PULL
1760   NEXT
1770   IF LEVEL<>REG THEN LEVEL=LEVEL+1 : GOTO *MAIN
1780   IF SP=0 THEN RETURN ←── プログラムを終了するかの判定
1790 GOTO *MAIN
1800 '
1810 *PUSH
1820  STACK(SP)=P : STACK(SP+1)=Q           ┐
1830  STACK(SP+2)=V : STACK(SP+3)=E         ├ スタックに5つの主要データを退避させる
1840  STACK(SP+4)=LEVEL : SP=SP+5           ┘
1850 RETURN
1860 '
1870 *PULL
1880  IF SP<1 THEN SP=0 : RETURN                ┐
1890  SP=SP-5 : LEVEL=STACK(SP+4)               │
1900  VECTOR=STACK(SP+2) : E=STACK(SP+3)        ├ 前に退避したデータを復帰させる
1910  X=STACK(SP) : Y=STACK(SP+1)               ┘
1920 RETURN
1930 '
1940 *PENTAGON ←── 枝を構成する五角形を描く
1950  POINT(X,Y)
1960  IF SP/5<10 THEN C=3 ELSE C=4   ┐
1970  IF SP/5< 8 THEN C=5            │
1980  IF SP/5< 6 THEN C=6            ├ 五角形の色の決定
1990  IF SP/5< 4 THEN C=2            │
2000  IF SP/5< 2 THEN C=1            ┘
2010  FOR J=0 TO 3
2020    LINE -STEP(MATRIX(VECTOR,0,J)*E,-MATRIX(VECTOR,1,J)*E),7
2030  NEXT
2040  LINE -(X,Y),7
2050  IF SP/5<9 THEN PAINT(X+PM[VECTOR,0]*E,Y-PM[VECTOR,1]*E),7,7
2060  POINT(X,Y)
2070  FOR J=0 TO 3
2080    LINE -STEP(MATRIX(VECTOR,0,J)*E,-MATRIX(VECTOR,1,J)*E),C
2090  NEXT
2100  LINE -(X,Y),C
2110  IF SP/5<9 THEN PAINT(X+PM[VECTOR,0]*E,Y-PM[VECTOR,1]*E),C,C
2120 RETURN
2130 '
2140 *TRAP
2150  RESUME NEXT
2160 '
2170 *ARRAY
2180  DATA  0 , 1 , .5, .5, .5,-.5, 0 ,-1   ┐
2190  DATA  2 , 2 , 2 , 0 , 0 ,-2 ,-2 ,-2   │
2200  DATA  1 , 0 , .5,-.5,-.5,-.5,-1 , 0   │
2210  DATA  2 ,-2 , 0 ,-2 ,-2 , 0 ,-2 , 2   ├ 8つの方向それぞれの
2220  DATA  0 ,-1 ,-.5,-.5,-.5, .5, 0 , 1   │ 五角形の素となるデータ
2230  DATA -2 ,-2 ,-2 , 0 , 0 , 2 , 2 , 2   │
2240  DATA -1 , 0 ,-.5, .5, .5, .5, 1 , 0   │
2250  DATA -2 , 2 , 0 , 2 , 2 , 0 , 2 ,-2   ┘
2260 '
2270 *ARRAY1
2280  DATA  .5, .5, 2 , 0 , .5,-.5, 0 ,-2   ┐ 8つの方向それぞれの
2290  DATA -.5,-.5,-2 , 0 ,-.5, .5, 0 , 2   ┘ 五角形の中の塗り始めを決定するデータ
```

1本の枝を描き終えたところ

LENGHT, LEVEL 共に4の場合の完成図形

7章　動的グラフィックス

7.1　パターンの移動

今までの章からも分かるように，PC-8801のグラフィックスの機能は，時間と手間さえ惜しまなければ，かなり美しいパターンや複雑な素材を扱うことができます．しかし，この上〝動き〟というものを取り入れた時，はたして〝動き〟に感じられるかどうかは不安です．動いて見えるということは，短い時間（人間が気付かないくらい）のうちに，ある物が他の物に変化する——この場合，形や位置の変化のことですが——そういったことが必要となります．実際，画面全体を用いてアニメーションを実現させようなどと考えてみても，８ビットのマイクロコンピュータである以上，スピードの点で不可能と言えるでしょう．ですから，画面全体でアニメーションを行うことをあきらめなければならないようです．そこでこの章では，単純で小さなパターンを対象に，アニメーションらしきことをしようという訳です．

では，何故，単純で小さなパターンだと，アニメーションができるのでしょう．何かメリットがあるのでしょうか．メリットを考えるには，アニメーションを実現する為の，アルゴリズムを考えてみるのが早道だと思われます．

今，よく目にするアニメーション——いわゆる，テレビや映画のアニメのたぐいですが——は，セルという透明のシートに，少しずつずれた絵を何千枚，何万枚も描き，それを１コマから数コマずつフィルムに撮影します．そして，出来上がったフィルムを続けて見ると，バラバラであったセルの絵が，動画（アニメーション）として，スクリーン上に再生される訳です．この様な方法をそのまま，コンピュータで実現するには，高速の処理を可能にするシステムが必要となりそうです．グラフィック命令の処理があまり早くなく，フィルムの様に多くの画面(白黒モードで，３つのバンクを独立させて用いれば，３枚の画面が得られたことになりますが，今は，それより多い枚数の画面を考えることにします）を持ち合わせていないPC-8801では，違う方法によらなければならないのです．

では，簡単なアニメーションのアルゴリズムを考えてみましょう．

［Ｉ］ ① 何か，パターンを描く．
　　 ② そのパターンを消す．
　　 ③ 次のパターン（①と少し異なったパターン）を描く．
　　 ④ ②の作業に戻る．

［Ｉ］に示したのが，そのアルゴリズムです．この流れを繰り返すことによって，アニメーションは実現できる筈です．さて，ここで話を元に戻して，単純で小さなパターンのメリットについて考えてみましょう．

単純で小さなパターンを用いるというのは，上記の流れの中では，①や③の〝描く〟という作業を早くし，かつ②の〝消す〟作業をも，早く処理できる様にします．要は，早く処理することが，アニメーションを実現する上での最大のメリットなのですから，PC-8801で実現するには，単純で小さなパターンを用いて，少しでも早い処理を得ることがメリットにつながるのです．では，物理で御馴染みの放物運動を，先程示したアルゴリズムを用いてシミュレートしてみましょう．

シミュレーションのプログラムは，リスト１に示しました．最初の入力文で，初速 V_0（m/sec）と投げ出し角度 θ（度），記録される時間（秒）とその刻み時間（秒）を与えます．例として，リスト１の初めのコメントに，比較的良い値（といっても，ただ単にプログラムに順応しているだけですが……）を示しておきました．この与えられた数値を用いれば，放物運動する際の位置は，次の式によって求めることができます．

$$\left.\begin{array}{l} \text{X方向の位置}：V_0 \cdot \cos\theta \cdot t \\ \text{Y方向の位置}：V_0 \cdot \sin\theta \cdot t - \frac{1}{2} \cdot g \cdot t^2 \end{array}\right\} \quad \cdots\cdots (1)$$

但し，gは重力加速度で9.8（m/sec²）
　　tは経過した時間（秒）

リスト１

```
1000 '
1010 '  ball demonstration by circle
1020 '
1030 '   V0     = 100   P  = 45
1040 '   T.MAX = 16    DT = .5
1050 '
1060  CLS
1070  INPUT "ｼｮｿｸ     V0    : ",V0
1080  INPUT "ｶｸﾄﾞ     P     : ",P
1090  INPUT "ｷﾛｸｼﾞｶﾝ  T.MAX : ",T.MAX
1100  INPUT "ｷｻﾞﾐ     DT    : ",DT
1110  G=9.8
1120 '
1130  DIM XP(INT(T.MAX/DT)),YP(INT(T.MAX/DT))
1140  SCREEN 0,1 : CLS 3
1150  FOR T=0 TO T.MAX STEP DT
1160    GOSUB *EQUATION
1170  NEXT
1180 '
1190  MAX.X=XP(0) : MAX.Y=YP(0)
1200  MIN.X=XP(0) : MIN.Y=YP(0)
1210  FOR I=1 TO INT(T.MAX/DT)
1220    IF MAX.X<XP(I) THEN MAX.X=XP(I)
1230    IF MAX.Y<YP(I) THEN MAX.Y=YP(I)
```

Time = 6.0 second.

```
1240     IF MIN.X>XP(I) THEN MIN.X=XP(I)
1250     IF MIN.Y>YP(I) THEN MIN.Y=YP(I)
1260   NEXT
1270 '
1280   SCREEN 1,0,1 : GOSUB *WINDOW.SET
1290   T.P=0
1300   CIRCLE(XP(0),YP(0)),10
1310   FOR T=1 TO INT(T.MAX/DT)-1
1320     T.P=T.P+DT
1330     LOCATE 0,0 : PRINT USING "Time = ###.# second.";T.P
1340     CIRCLE(XP(T-1),-YP(T-1)),10,0 ←── 1回前に描いた円を消す
1350     CIRCLE(XP(T),-YP(T)),10  ←──── 新しく円を次の位置に描く
1360   NEXT
1370 END
1380 '
1390 *EQUATION
1400  XP(INT(T/DT))=V0*COS(P*3.14159/180)*T ← 円の位置計算をする
1410  YP(INT(T/DT))=V0*SIN(P*3.14159/180)*T-G*T^2/2
1420 RETURN
1430 '
1440 *WINDOW.SET
1450  WINDOW(MIN.X-50,-MAX.Y-50)-(MAX.X+50,-MIN.Y+50)
1460  VIEW(0,0)-(639,199)       ←─ 表示範囲の決定 ↗
1470 RETURN
```

$v_0\sin\theta$　v_0　θ　$v_0\cos\theta$

図1　初速V_0角度θで投げた時のx，y各方向の成分

ちょっと物理学の話を交えたので，難しいと感じられた人もいると思いますが，この章は，物理学の章でなく動的シミュレーションの章ですから，この式を用いれば，t秒後の位置が計算できるということだけ知っておいてください(参考のために，図1に，初速V_0と角度θから，x，yそれぞれの方向の初速は，$V_0\cos\theta$，$V_0\sin\theta$であることを示しました).

さて，動くパターンは，〝単純〟という条件を満たすために小さな円を用いてありますが，これを画面上で描いたり消したりする度に，与えられた様々な値から，式（1）によってその位置を計算していたのでは，早さを追求する上で多少不利になります．そこで，一箇所でまとめて位置の計算をしておいて，小さな円を描く時には，その計算しておいた結果のみを参照するようにしてみます．リスト1では，3つのFOR～NEXTループを含んでいますが，その第1番目のループで，すべての位置の座標を計算して配列内に記憶させてしまいます．位置のx座標の値を格納しておく配列が，XP（ ）で，y座標の値を格納しておく配列が，YP（ ）です．後で参照するのは，この2つの配列のみになり，計算の手間は軽減されました．式（1）を計算するルーチンが，*EQUATIONです．従って，このサブルーチンでもっと複雑な計算（今考えている放物運動は，空気抵抗やその他諸々の事象は全く影響しないもので，単純な運動です）をすれば，それなりの運動のシミュレーションを行えます．

2つ目のFOR～NEXTループでは，計算して得た位置のx方向，y方向それぞれの最大値と最小値を求めます．この値を用いて，PC-8801の持っている機能であるWINDOW文を有効に使

(a) リスト1, 2, 3におけるウィンドウの切り方　(b) yの値を－1倍しない場合　(c) yの値を－1倍した場合

図2

って，表示域を画面一杯になるように設定します．WINDOW 文によって，これを実行しているサブルーチンが，＊WINDOW．SET で，実際の値よりひとまわり大きくウィンドウを切るようにしています．y 方向の値は，負にすることによって，表示する位置と求めた値のつじつまを合わせています．つまり，図2(a)の様にウィンドウを切った時，上に向かうほどy座標の値が小さくなるのに反して，計算で求めた結果というのは，高い位置ほど大きな値を持っていますので，結果を負の値に変換しないでそのまま表示させたとすれば，その放物線は図2(b)の様になってしまいます．

従って，図2(c)の様に正しく表示させるには，x軸に対称な形のものを画面に出力しなければならないので，いつもの通りy方向の値を負の値に変換（－1倍）します．この考えは，3つ目の FOR～NEXT ループ内の小さい円の表示や消去の際にも用いています．

では，いよいよ本題である〝動画〟を実現している，3つ目のループについて説明しましょう．

このループの流れを考える時に，思い出してもらいたいのが，[Ｉ]で示したことです．まず，ループに入る前にある CIRCLE 文が，[Ｉ]の①に当たり，何かのパターンというのが，この場合，円になっています．次に続くループが，[Ｉ]の②～④の一連の流れに相当しており，②のパターンを消すという操作は，黒の円を前のパターン（円）に重ねて描くことに相当します．そして，黒で円を描いた後で，次の位置に円を描いています．これは，[Ｉ]の③に相当します．

以上の説明から分かる様に，リスト1に示したプログラムは，忠実に[Ｉ]に示されたアルゴリズムをたどっています．

では，全く同じアルゴリズムで，②のパターンを消す操作を，LINE 文の BF オプションを用いて実行してみましょう．

〝パターンを消す〟ということを，画面全体を消すことで代用するのは，機械語のパッケージでも使わない限り，当然ながら馬鹿げています．そこで，画面の長方形の一部を消去するために，黒で LINE 文の BF オプションを実行してみます．その方法によって作ったプログラムが，リスト2に示したものです．

リスト2

```
1000 '
1010 '   ball demonstration by line ; box fill
1020 '
1030 '    V0     = 100    P  = 45
1040 '    T.MAX = 16     DT = .5
1050 '
1060  CLS
1070  INPUT "ｼｮｿｸ     V0     : ",V0
1080  INPUT "ｶｸﾄﾞ     P      : ",P
1090  INPUT "ｷﾛｸｼﾞｶﾝ  T.MAX  : ",T.MAX
1100  INPUT "ｷｻﾞﾐ     DT     : ",DT
1110  G=9.8
1120 '
1130  DIM XP(INT(T.MAX/DT)),YP(INT(T.MAX/DT))
1140  SCREEN 0,1 : CLS 3
1150  FOR T=0 TO T.MAX STEP DT
1160    GOSUB *EQUATION
1170  NEXT
1180 '
1190  MAX.X=XP(0) : MAX.Y=YP(0)
1200  MIN.X=XP(0) : MIN.Y=YP(0)
1210  FOR I=1 TO INT(T.MAX/DT)
1220    IF MAX.X<XP(I) THEN MAX.X=XP(I)
1230    IF MAX.Y<YP(I) THEN MAX.Y=YP(I)
1240    IF MIN.X>XP(I) THEN MIN.X=XP(I)
1250    IF MIN.Y>YP(I) THEN MIN.Y=YP(I)
1260  NEXT
1270 '
1280  SCREEN 1,0,1 : GOSUB *WINDOW.SET
1290  T.P=0
1300  CIRCLE(XP(0),YP(0)),10
1310  FOR T=1 TO INT(T.MAX/DT)-1
1320    T.P=T.P+DT
1330    LOCATE 0,0 : PRINT USING "Time = ###.# second.";T.P
1340    LINE(XP(T-1)-20,-YP(T-1)-10)-STEP(40,20),0,BF   ← 1回前に描いた円を消す
1350    CIRCLE(XP(T),-YP(T)),10   ← 次の位置に円を描く
1360  NEXT
1370 END
1380 '
1390 *EQUATION
1400  XP(INT(T/DT))=V0*COS(P*3.14159/180)*T          } 円の位置計算
1410  YP(INT(T/DT))=V0*SIN(P*3.14159/180)*T-G*T^2/2  }
1420 RETURN
1430 '
1440 *WINDOW.SET
1450  WINDOW(MIN.X-50,-MAX.Y-50)-(MAX.X+50,-MIN.Y+50)  } 
1460  VIEW(0,0)-(639,199) ←―――――――― 表示範囲の決定    }
1470 RETURN
```

リスト1と見較べれば1目瞭然ですが，アルゴリズム[I]の②に挙げてある，パターン消去に対応する行が変っただけです．すなわち，リスト1では，

```
1340 CIRCLE(XP(T-1),-YP(T-1),10,0
```

によって，パターン消去を行っている所を，リスト2では，

```
1340 LINE(XP(T-1)-20,-YP(T-1)-10)-STEP(40,20),0,BF
```

と書き換えてあります．その他の部分は，すべてリスト1のプログラムと同じものなので，リスト2についての説明はこれまでにしておきます．

さて，[I]のアルゴリズムに従った，2つのプログラムを見てきましたが，どうも，パターンの消去や描き込みを実行しているのが画面に出てしまって，アニメーションという雰囲気を一味

欠いているように思われます．せめて，消している画面を表に出さない工夫ぐらいはしたいものです．そこで，考えられるアルゴリズムが，次に示すものです．

［II］ ① あるパターンを描く．
② 画面への出力を禁止する．
③ ①で描いたパターンを消去する．
④ 次のパターン（①と少し異ったもの）を描く．
⑤ 画面への出力を有効にする．
⑥ ②の作業に戻る．

このアルゴリズムで［I］と異なっている部分は，②の〝画面への出力を禁止する〟と⑤の〝画面への出力を有効にする〟という2つのステップだけですから，アニメーションが［I］のアルゴリズムで実現されている以上，［II］でも同様に動作する筈です．しかし，よく考えてみると，②から⑤の間――すなわち，画面に何も出ていない間に，2つのステップをこなすのは，多少長い時間を費やす様な気もします．となると，やはり不自然な動作になるのでしょうか．もし，そうだとしたならば，自然に動かし，かつ，消去を見せない工夫はあるのでしょうか．

ところで，PC-8801には，白黒モードの際3つのページが存在します．このページを用いて，表示しているページと消去・描き込みを行っているページという様な分担作業をすれば，常に表示しながら表示していないページに対して，消去・描き込みができます．では，このことを使ってアルゴリズムを考え直してみましょう．

先程も述べた様に，ページを分担作業になる様にして，有効に用いることを考え，〝表示しているページ〟と〝消去・描き込みをするページ〟の2つの機能に分けます．そして，2枚のページが，この2つの機能を交互に受け持つことを繰り返せば，どうやらアニメーションがうまくできそうです．そこで，今までの考え方をまとめてみると，以下に示すようになります．

［III］ ① 表示されていないページにパターンを描く．
② 今まで，表示されていたページと表示されていなかったページの機能を入れ換える．従って，表示されていなかったページ（①でパターンを描いたページ）が表示される．
③ 表示されていたページに描いてあったパターンを消去する．但し，1回目には，このプログラムは，何も描いていない所を消すので無意味になる．
④ ①の作業へ戻る．

このアルゴリズムを用いて，前と同様に放物運動をアニメーションによってシミュレートしたプログラムが，リスト3です．リスト1，2のプログラムと異なる点は，アニメーションを行う3つ目のFOR～NEXTループです．このループの中において，第1番目のSCREEN文とCIRCLE文の2行は，アルゴリズム［III］の①に当る部分です．次に続くSCREEN文は，②の操作に当たり，更に次のCIRCLE文で③の部分を処理しています．FOR～NEXTループに入る直前に，変

数A，Bに対する代入文がありますが，それぞれの変数は，ループの内部で表示されている画面と表示されずに，パターンの消去・描き込みがされている画面の入れ換えをする上で，重要な役割を果たしています．

リスト3

```
1000 '
1010 '   ball demonstration by paging
1020 '
1030 '    V0    = 100    P  = 45
1040 '    T.MAX = 16     DT = .5
1050 '
1060  CLS
1070  INPUT "ｼｮｿｸ     V0    : ",V0
1080  INPUT "ｶｸﾄﾞ     P     : ",P
1090  INPUT "ｷﾛｸｼﾞｶﾝ  T.MAX : ",T.MAX
1100  INPUT "ｷｻﾞﾐ     DT    : ",DT
1110  G=9.8
1120 '
1130  DIM XP(INT(T.MAX/DT)),YP(INT(T.MAX/DT))
1140  SCREEN 0,1 : CLS 3
1150  FOR T=0 TO T.MAX STEP DT
1160    GOSUB *EQUATION
1170  NEXT
1180 '
1190  MAX.X=XP(0) : MAX.Y=YP(0)
1200  MIN.X=XP(0) : MIN.Y=YP(0)
1210  FOR I=1 TO INT(T.MAX/DT)
1220    IF MAX.X<XP(I) THEN MAX.X=XP(I)
1230    IF MAX.Y<YP(I) THEN MAX.Y=YP(I)
1240    IF MIN.X>XP(I) THEN MIN.X=XP(I)
1250    IF MIN.Y>YP(I) THEN MIN.Y=YP(I)
1260  NEXT
1270 '
1280  SCREEN 1,0,1 : GOSUB *WINDOW.SET
1290  A=0 : B=1 : T.P=0
1300  FOR T=1 TO INT(T.MAX/DT)-1
1310    SWAP A,B ←── 2つのページの機能を入れ換える
1320    T.P=T.P+DT
1330    LOCATE 0,0 : PRINT USING "Time = ###.# second.";T.P
1340    SCREEN ,,B,A+1 ←── 表示されているページとされていないページを入れ換える
1350    CIRCLE(XP(T),-YP(T)),10 ←── 表示されていないページに円を描く
1360    SCREEN ,,A,B+1 ←── 表示されていないページとされているページを入れ換える
1370    CIRCLE(XP(T-1),-YP(T-1)),10,0 ↖
1380  NEXT                        表示されていないページの１回前に描いた円を消す
1390 END
1400 '
1410 *EQUATION
1420  XP(INT(T/DT))=V0*COS(P*3.14159/180)*T
1430  YP(INT(T/DT))=V0*SIN(P*3.14159/180)*T-G*T^2/2 ↖
1440 RETURN                                          円の位置計算
1450 '
1460 *WINDOW.SET
1470  WINDOW(MIN.X-50,-MAX.Y-50)-(MAX.X+50,-MIN.Y+50)
1480  VIEW(0,0)-(639,199)                          表示範囲の決定
1490 RETURN
```

ここで，SCREEN 文のパラメータの意味を思い出してください．今このループに入る時点で，グラフィック画面のモードは白黒モードになっており，３ページあるプレーンのうち，２ページ目を表示しています．そして，ループに入って最初のSCREEN 文では，第３パラメータにB，第４パラメータにA＋1が置かれていますから，その意味は，『変数Bの示すページを表示し，変数Aの値が意味する所のページに対して，グラフィック命令を有効にする』ということになります．ですから，ループに入ってすぐの状態では，A＝1，B＝0なので，『1ページ目を表示し，2ページ目を有効にする』ことになり，この場合，1ページ目を表示しながら，円が描かれるのは2

ページ目になるのです．そして，2番目のSCREEN文も，同じ様に考えれば2ページ目を表示し，1ページ目を有効な画面にするので，その前の行で描いた円が画面に現れます．この時，有効なページは今まで表示していたページだった訳ですから，ループ最後のCIRCLE文は，画面を表示しない状態で前に描いた円を消去します．この様な流れをループに入ってすぐのSWAP文で，変数AとBの内容を入れ換えながら実行しているので，結局，画面の表示ページと有効ページの入れ換えをしているのと同じことになります．

さて，アニメーションの原理らしきものを，3つの例によって説明しましたが，大体分かって頂けたでしょうか．ページを用いたアニメーションは，7．3にも取り上げてありますが，何しろ，PC-8801では，白黒モードで3ページしか得ることができません．よって，早い動きを得るには，前々から述べているように，単純なパターン（単色という意味も含みます）にならざるを得ない訳です．PC-8801のBASICのみを用いて正統な考え方でアニメーションを行うことは，繰り返して言うようですが，スピードの点で多少問題があり，無謀ともいえるかもしれません．この節では，それを克服するのに，様々な工夫が必要になってくることに気付いてくだされば十分です．

7.2 カラー・パレットの利用

前節でも明らかになった様に，アニメーションを実現するには，数多くの困難が待ち構えています．PC-8801の持っているグラフィックの能力は，8ビットのマイクロコンピュータ，その上インタプリタ系の言語（BASICなど）でサポートするには，グラフィック専用のプロセッサでも備えていない限り，その処理スピードに対して期待を持つことはあまりできない様です．従って，何か工夫（悪く言ってしまえば，〝ごまかし〟ということにもなりかねませんが……）をすることによって，如何にも早く処理している様に見せかける必要も生じてきます．この節では，現在ある8ビットのコンピュータの中では，PC-8801特有の機能である，パレットを用いることによって，それを〝工夫〟の手助けとし，アニメーションにチャレンジしてみます．

パレットの概念などの詳細について，ここで繰り返すことはやめておきます．今は，N_{88}-BASICにおける，パレットをコントロールする命令，

COLOR＝(パレット番号，カラー・コード)

のみ理解していれば問題ありません．

もし，今，手元にPC-8801がある方は，ダイレクト・モードで，

```
SCREEN 0,0:LINE(0,0)-(639,199),4,BF
```

としてみてください．画面は，パレットが普通の状態なら緑色になった筈です．そこで，もう一度ダイレクト・モードで，

```
COLOR=(4,2)
```

とすれば，画面は一瞬のうちに赤に変わったと思います．さらに，

```
COLOR=(4,4)
```

とすれば，再び，一瞬のうちに緑の画面に戻ります．ここで大切なのは〝一瞬のうちに〟変化することであり，これを利用すれば，早い動きを得られるかもしれないことに，誰しも気付くことでしょう．但し，パレットの数に限界がありますから(PC-8801 の場合は 8 個)，動きのダイナミックさにも影響してきます．従って，同じ場所に描かれたパターンを非周期的，もしくは，周期的に何回も使用するようなプログラムにならざるを得ません．このことを考えると，カラー・コードの種類と同数のパレットでなく，もっとたくさんのパレットを用意しておいてくれると有難かったのですが，今となっては，この不満を取り除くことはできないので目をつぶることにしましょう．

リスト4

```
1000 '
1010 '   ﾊﾟﾚｯﾄ ﾆ ﾖﾙ ｾﾝｺｳﾊﾅﾋﾞ ﾉ ﾃﾞﾓﾝｽﾄﾚｰｼｮﾝ
1020 '
1030  SCREEN 0,0 : CLS 3
1040  WINDOW(-100,-100)-(100,100)
1050  VIEW(0,0)-(639,199)
1060 '
1070  FOR C=1 TO 7                                  ┐
1080    READ CX,CY                                  │
1090    FOR I=0 TO 6 STEP .05                       │
1100      X=30*RND*COS(I) : Y=30*RND*SIN(I)         ├ 火花を描く
1110      LINE(CX,CY)-(CX+X,CY+Y),C                 │
1120    NEXT                                        │
1130  NEXT                                          ┘
1140 '
1150  FOR I=0 TO 7      ┐
1160    COLOR=(I,0)     ├ 火花の表示をなくす
1170  NEXT              ┘
1180 '
1190  FOR V=0 TO 13                                 ┐
1200    IF (F AND 1)=0 THEN C=3 ELSE C=4            │
1210    F=F XOR 1                                   │
1220    COLOR C : LOCATE 40,V : PRINT "|"           ├ 線香花火を描く
1230  NEXT                                          │
1240  COLOR 2 : LOCATE 40,V : PRINT "●"             ┘
1250 '
1260  MAX=100
1270  FOR L=0 TO 300
1280    C=INT(RND*7+1)
1290    COLOR=(C,6) ←── 選ばれたパレット番号の火花の表示
1300    FOR D=0 TO MAX : NEXT ←── インターバル
1310    BEEP 1 : BEEP 0
1320    COLOR=(C,0) ←── 表示した火花の消去
1330    MAX=MAX-1
1340  NEXT
1350 '
1360  FOR I=14 TO 24                    ┐
1370    LOCATE 40,I : PRINT " ";        │
1380    COLOR 2                         │ 線香花火の火の玉が
1390    LOCATE 40,I+1 : PRINT "●";      ├ 落ちるアニメーション
1400    FOR D=0 TO 50 : NEXT            │
1410  NEXT                              │
1420  LOCATE 40,24 : PRINT " ";         ┘
1430 '
1440  IF INKEY$="" THEN 1440 ←── キーの入力待ち
```

```
1450  FOR I=0 TO 7
1460    COLOR=(I,I)    ←── パレットを正常状態にする
1470  NEXT
1480 END
1490 '
1500  DATA -40,-40,-40,20,-10,-10,40,40
1510  DATA 10,60,50,-10,40,-50
```

リスト4に示したプログラムは，パレットを用いてアニメーションを行うもので，同じパターンを非周期的に何回も使用するタイプのものです．

プログラムの最初の FOR～NEXT ループで，ランダムな長さの放射状の図形を，黒以外の色で描きます．この時，放射し始める元となる点の座標は，既にデータ文で与えてあります．

2番目のループは，7個の放射図形の色をパレットを操作することによって黒にし，見えなくします．そして，この後に続く6行で，キャラクタ画面を用いてアニメーションの演出をします．実行すれば分かりますが，この演出は，『線香花火』に他なりません．要するに，放射状の図形は，その花火のデモンストレーション用のもので，これをパレットを用いて時々表示し，花火が散っている様子を表現します．

アニメーションの核となる部分は，4番目の FOR～NEXT ループで変数Lの上限値が，火花の散り終わりまでの時間を決定しており，変数 MAX が，1回火花が散って次に散るまでの周期を決定します．ループの中で，変数 MAX の値は1ずつ小さくなっていくので，火花の散る周期は本物の線香花火のように後になれば後になるほど，短くなります．どのパレット番号に対応する火花を表示するかは，このプログラムでは，乱数を用いて1～7の整数をでたらめに発生させることによって，決めています．乱数によって決定された番号（変数Cが保持している）の火花は，

```
COLOR=(C,6)
```

の文で，黄色く表示されます．そして，変数 MAX を用いた FOR～NEXT ループでインターバルを取り，次の文，

```
COLOR=(C,0)
```

で，元通り黒にします．これらの流れを変数Lで示された回数分，変数MAXの値を1ずつデクリメントしながら繰り返します．

5番目のFOR～NEXTループも，キャラクタによるグラフィックの演出です．こちらの方は，先程のものと違い，線香花火の火の玉が落ちるのを，キャラクタによるアニメーションを使って表しています．プログラムは，アニメーションの部分を終了すると，キー入力が何かあるまで待っていますから，スペース・バーでも入力してください．パレットが正常な状態に戻されて，プログラムが終了します．

ここで注目して欲しいのは，このプログラムにおけるアニメーションのスピードです．パレットを用いたことによって，BASICだけ（機械語に頼っていない）で記述されたプログラムが実行されているとは思えないくらい早い動きが得られました．ですから，アニメーションに必要であった，〝スピード〟感という目的は，少なくとも達成された訳です．

さて，リスト4のプログラムでは，異なった色のパターンを非周期的に何回も使用するタイプでしたが，今度は，周期的な動きをするアニメーションを，パレットによって実現してみたのが，リスト5に示してあるプログラムです．

リスト5

```
1000 '
1010 '   animation by palette rotation
1020 '
1030 DIM G1(1,50),G2(1,50),DX(50),DY(50),X(50),Y(50)
1040 '
1050 SCREEN 0,0 : CLS 3
1060 INPUT "How many points";N
1070 '
1080 CLS
1090 LINE(320,0)-(320,199),4  }座標軸を描く
1100 LINE(0,100)-(639,100),4  }
1110 '
1120 PRINT "First shape -"  ←── 初めの状態を座標で入力
1130 FOR I=1 TO N
1140   INPUT "(x,y) : ",G1(0,I),G1(1,I)
1150   IF I<>1 THEN LINE(320+G1(0,I-1)/.4,100-G1(1,I-1))-(320+G1(0,I)
/.4,100-G1(1,I)),1
1160 NEXT
1170 '
1180 PRINT "Last shape -"  ←── 最後の状態を座標で入力
1190 FOR I=1 TO N
1200   INPUT "(x,y) : ",G2(0,I),G2(1,I)
1210   IF I<>1 THEN LINE(320+G2(0,I-1)/.4,100-G2(1,I-1))-(320+G2(0,I)
/.4,100-G2(1,I)),2
1220 NEXT
1230 '
1240 PRINT "pattern making !"
1250 FOR I=1 TO N                 }作成する7つのパターンの隣り合った2つの
1260   DX(I)=(G2(0,I)-G1(0,I))/7  }x方向，y方向それぞれの差を計算
1270   DY(I)=(G2(1,I)-G1(1,I))/7  }
1280 NEXT
1290 '
1300 CLS 3
1310 FOR I=1 TO 7
1320   FOR J=1 TO N
1330     X(J)=320+(G1(0,J)+DX(J)*I)/.4           }
1340     Y(J)=100-(G1(1,J)+DY(J)*I)              }上のFOR～NEXTループで計算した
1350   NEXT                                      }値を用いて7つのパターンを作る
1360   FOR J=1 TO N-1                            }
1370     LINE(X(J),Y(J))-(X(J+1),Y(J+1)),I       }
```

```
1380     NEXT
1390  'NEXT
1400  '
1410   FOR I=0 TO 7   ｝
1420     COLOR=(I,0)  ｝パターンの表示をなくす
1430  'NEXT          ｝
1440  '
1450   FOR G=0 TO 10
1460     FOR J=1 TO 7
1470       FOR I=1 TO 7
1480         COLOR=(I,J)←── パターンの表示
1490         GOSUB *DELAY
1500         COLOR=(I,0)←── 表示したパターンの消去
1510       NEXT
1520       GOSUB *DELAY
1530       FOR I=7 TO 1 STEP -1
1540         COLOR=(I,J)←── パターンの表示
1550         GOSUB *DELAY
1560         COLOR=(I,0)←── 表示したパターンの消去
1570       NEXT
1580     NEXT
1590   NEXT
1600 END
1610 '
1620 *DELAY
1630   FOR I.D=0 TO 50 : NEXT
1640 RETURN
```

プログラムの概要を説明しますと，任意の数の点を結んでできる線画を２つ与えた時，初めに与えた図形から２番目に与えた図形への変化の過程を７種類の図形に分割し，それらを，次々と表示するのです．例えば，図３(a)の様な３点を結んでできる図形と，図３(b)の様な図形を与えますと，図３(c)の様な７種類の図形に分割します．この分割されたそれぞれに，順に１～７の番号を振ってみましょう．そして，１→２→……→６→７→６→……→２→１→２→……の様に周期的に選んで，選ばれた図形を表示するようにしますと，１つのアニメーションが出来上がります．ここで，振り付けた番号をパレット番号と考えると，数値を周期的に選ぶことがパレットを選択することになり，同時に表示する図形を選ぶことにもなります．

(a)　　(b)　　(c)

図３　リスト５の図形の作成の仕方

プログラムの流れについて説明する前に，まず各配列の役割を紹介しておきましょう．

Ｇ１(　)　：**初めに与えるパターンの座標を保持します．**

Ｇ２(　)　：**次に与えるパターン（終わりのパターン）の座標を保持します．**

ＤＸ(　)　：**パターンの各点について，あるパターンと次のパターンのｘ座標の差を格納します．**

DY(　)　：DX(　)と同様に，y座標の差を格納します．

X(　)　：DX(　)によって得る，変化の過程のパターンのx座標を，一時的に持っているワーク・エリアです．

Y(　)　：X(　)と同様に，変化の過程のパターンのy座標を，一時的に持っているワーク・エリアです．

では，例を挙げて，プログラム中における配列の使われ方を考えてみます．

DIM 文の後の，INPUT 文で〝2〟と答えたとしましょう．これで図形は2点から成り立つものと解釈されます．次に画面上に目盛が表示されてから，

```
First shape  —
```

と表示され，座標入力を促してきますので，座標を入力していき，それと同時に図形を描いていきます．この時入力する座標が，配列G1に格納されます．すべての点（今は，2点）の座標の入力が終了すると，

```
Last shape  —
```

と表示され，再び，座標入力を促してきますので，最初の時と同様にしてください．この時の座標は，配列G2の方に格納されていきます．こちらの入力を終えると，今度は，

```
Pattern making  !
```

の表示があり，リスト5でこの処理をしている部分を見ると，配列 DX と DY に，各点の最初と最後の座標の差の7等分の値（x軸方向，y軸方向について，それぞれの差の7等分の値）を代入しています．7等分しているのは，0番以外の7個のパレットを用いて，7種類の図形を描くためです．従って，最初に与えたパターンの各点の座標に，ここで求めた各点ごとの座標の増分を順に加えていけば，7つの図形の各点の座標が得られます．その7つの図形1つ1つを描く際に，ある1つの図形の座標の並びを一時的に保持しておく配列が，XとYで，リスト5では，4つ目の FOR～NEXT ループで，あるパレット番号（変数Iが示す）の時の各点の座標を計算しながら代入していって，その後，LINE 文で図形を描きます．

以上で，アニメーションの下準備ができたことになります．配列の用いられ方は，理解できたでしょうか．

7つの図形を描き終えたあとの FOR～NEXT ループは，リスト4の時と同じく，画面の図形の色をすべて黒にして，表示されているのが分からない様にします．後は，アニメーションをするために，周期的にパレット番号を選び，それを，

```
COLOR=(I,J)
```

で表示するだけです．変数Ｉが，表示される図形のパレット番号を示し，表示される色が変数Ｊで決まります．また，くどい様ですが，移動を含むアニメーションは，前に描いた図形を消してから次の図形を描かなければなりませんから，必ず，

```
COLOR=(I,0)
```

によって，表示したパターンを消しておきます．

リスト４の例でも分かる通り，パレットを用いたアニメーションは，時として早過ぎると思われるくらいのスピードで処理するので，リスト５では，＊DELAY というサブルーチンを設けて，少しだけゆっくりとしたアニメーションになる様に手を加えてあります．

この様に，最初の図形と最後の図形を与えて，その間の変化の過程を計算によって求める手法は，変態（メタモルフォーゼ：metamorphose）と呼ばれ，コンピュータ・アニメーションなどで使われています．

ここでまとめとして，パレットを用いたアニメーションにおける欠点をいくつか考えてみます．あまりに複雑な図形ですと，各パレットごとのパターンの交わる点は，後から描いた色になってしまうため，アニメーションの際に欠けた点を持つことになり望ましくありません．また，アニメーションを作成するときに用いることのできる図形の数が，パレットの数に限定されるということも問題です．スピードを得られてもその一方でこれら２つの問題が生じているので，やはり完璧と言えるアニメーションはなかなか望めないようです．

7.3 ページングの利用

次に試みる〝工夫〟は，PC-8801が持っている３つのグラフィック・バンク（プレーン）を利用してみます．これは，白黒モードの際に３ページ分のグラフィック画面しか得ることができませんから，当然単色のパターンのアニメーションにならざるを得ません．また，３ページしかないということは，３種類の図柄しか利用できないことですから，やはり多くの制限を受けることになります．

しかし，これら３ページは，相互に干渉することのない全く別のものですから，７．１の様にパターンを小さくする必要はなく，画面いっぱいの大きさまでなら，そのパターンはアニメーションに利用することができ，７．２の様にパターンが複雑になりすぎて，交差するとその点が欠けてしまって困るということなどからは，解放されることになります．

リスト６とリスト７に挙げたプログラムが，ページを利用したアニメーション――いわゆる，ページングによるアニメーションの例です．リスト７のプログラムは，リスト６に工夫を加えたもので，大まかなプログラムの流れはリスト６とほとんど同じです．何行かを付け加えるだけで，リスト７になりますから，順番通りリスト６のプログラムから説明していきます．

(a) ページ1　(b) ページ2　(c) ページ3

図4　リスト6のパターン作成の仕方

ページングによるアニメーションの考え方は，パレットの時と全く同じです．ただ，パレットの時に7種類のパターンが描けたのに対して，ページングの時には3種類のパターンしか描けません．ですから，この制約によって長い時間アニメーションを行うには，周期的な動きを必要とします．このプログラムでは，正六角形の各頂点上にある6つの円が，回転するデモンストレーションです．3つの各ページには図4のような図形が描かれます．

リスト6

```
100 '
110 '   demonstration of animation by paging
120 '
130  SCREEN 0,0 : CLS 3 : PRINT "Working !!"
140  FOR I=0 TO 2
150    SCREEN 1,1,I,2^I  ←── 描き込む画面(バンク)の選択
160    N=I*20
170    GOSUB *DRAW ←── 6個の円を描くサブルーチン
180  NEXT
190 '
200 *LOOP
210  FOR I=0 TO 2
220    GOSUB *WAITR
230    SCREEN ,,,2^I   ←── 3つの画面を順に表示
240  NEXT
250 GOTO *LOOP
260 '
270 *DRAW
280  WINDOW(-300,-180)-(300,180)
290  VIEW(0,0)-(639,199)
300  X=0 : Y=-100
310  SX=COS(2*3.1415/6)   : SY=SIN(2*3.1415/6)
320  SSX=COS(N*3.1415/90) : SSY=SIN(N*3.1415/90)
330 '
340  PP=X : X=SSX*X-SSY*Y
350  Y=SSY*PP+SSX*Y
360 '
370  FOR I.D=1 TO 6
380    P=X : X=SX*X-SY*Y
390    Y=SY*P+SX*Y
400    CIRCLE(X,Y),20,7 : PAINT(X,Y),7
410  NEXT
420 RETURN
430 '
440 *WAITR
450  FOR W=0 TO 100 : NEXT
460 RETURN
```

（340～350, 380～390行：$x = x\cdot\cos\theta - y\cdot\sin\theta$，$y = x\cdot\sin\theta + y\cdot\cos\theta$ の計算）

Working !!

y

(x', y')

(x, y)

θ

0

図5　回転移動のさせ方

正六角形の隣り合った2つの頂点と中心の点が作る角度は，図4(a)に示した様に60°です．別の考え方をすると，ある1つの頂点に着目した時，その点が隣の頂点に移動する――すなわち，全く同じ図形に戻るまでに60°回転するということですから，3枚のページに分けるには，図4の(b),(c)に示したように，60°÷3ページ＝20°ずつずれた図形を用意して，各ページに描き，それを順に表示すれば，回転運動のアニメーションになることが分かります．リスト6の＊DRAWというサブルーチンは，正六角形上の円を描きます．図5に示すように，任意の点の座標を（x，y）とした時，原点を中心に角度θだけ回転移動させた点の座標（x'，y'）は，

$$\left.\begin{aligned} x' &= x\cdot\cos\theta - y\cdot\sin\theta \\ y' &= x\cdot\sin\theta + y\cdot\cos\theta \end{aligned}\right\} \quad \cdots\cdots(1)$$

の式で与えられます（2．3．1〝円を描くには〟参照）．回転移動を開始する点の座標さえ決まれば，60°ずつ回転移動させて，正六角形の各頂点の座標が求められ，それぞれを中心に円を描けば，図4に示されるような図形が得られます．リスト6では，＊DRAWの中で2箇所に式(1)を用いて，計算を行っています．まず，FOR～NEXTループに入る前に，回転移動を開始する点の座標を決定するのに，式(1)を用いています．この時の回転する角度θは，変数Nが持っていて，0°，20°，40°の3つの場合があり，図4に示した3つの図にそれぞれ対応しています．変数SSXは，cos N，変数SSYは，sin Nの値が代入されており，それにより式(1)と同じ計算を行います．もう1箇所，式(1)は，θ＝60°のときのcosθを変数SXに，sinθを変数SYに代入し，FOR～NEXTループの中で用いられています．3種類の描き始める点に対して，それぞれSCREEN文によって，有効となる画面を変えながらパターンを描いていけば，図4と同様な図形が各ページに得られるのです．

後は，パレットを用いたアニメーションの時にパレット番号を選んでいた様に，ページ番号を選んで表示すれば，アニメーションになる筈です．ページを順序よく変える部分は，2番目のFOR～NEXTループで，アニメーションが早過ぎないように，サブルーチン＊WAITRで，遅れ(DELAY)をとっています．このスピードを落とすことは，パレットの時と同様な考慮で，サブルーチン＊WAITRがないと早過ぎて，表示の方が間に合わないということがありがちで，かえって美しい

アニメーションをそこないます.

しかし，リスト6のプログラムでは，回転運動といっても，6個の円が同時に回っています．やはり，回転運動らしくするには，1つの円が回っているのが良いのではないでしょうか．では，どうすれば1つの円の回転運動ができるかを考えてみましょう．

18ページ分のグラフィック画面をとることができれば，今までと全く同じ考え方で，20°ずつ回転移動させた位置に円を各ページ別に描いて，順に表示することで，1つの円が回転するようなアニメーションが可能ですが，残念ながらPC-8801では，最高3ページのグラフィック画面しか得ることができません．何か，他の〝工夫〟が必要となりそうです．

白黒のグラフィック画面では，ちょっと興味深い効果――すなわち，テキスト画面に対して有効となるべき，COLOR文やCOLOR＠文のテキスト・カラーを用いると，白黒のグラフィック・パターンに色を付けることができます．例えば，

```
CONSOLE ,,,1:SCREEN 1,0,0,1:COLOR 7
```

とした後，

```
LINE(0,0)-(639,199)
```

として白い線を描きます．この時，テキスト画面は，CONSOLE文でカラー・モードになっているものとします．次に，

```
COLOR 4:CLS
```

としてください．グラフィック画面に描いた筈の線にも，緑色になったと思います．続いて，次の文を入力してみてください．

```
COLOR@(0,0)-(20,20),2
```

画面の線の一部が赤くなったと思います．これを用いてリスト6のアニメーションを改良することによって，1つの円が回転運動するようにしてみましょう．

まず，不要な部分を黒い色を付けて消してしまい，次に，着目しているパターンのみCOLOR＠で，図6の様にして色を付けて表示します．この操作を，ページを換えながら6個の円について順に行えば，あたかも，1つの円が回転運動をしているように見えます．

リスト7では，COLOR＠で表示する部分の座標を＊COLOR．DATA以降のDATA文によって示してあります．そのデータは，最初のFOR～NEXTループで配列に格納していきます．ここで実際には，3画面で各6箇所を表示するために，それぞれ4つの値（表示する長方形の部分の左上と右下のキャラクタ座標）が必要で，72個（3×6×4）のデータがある筈なのに，プログラムでは，48個しかデータがないことに気付いた人もいると思います．このプログラムでは，6つの円のうちCOLOR＠文でどれを表示するかを示すキャラクタ座標のデータは，3画面のう

図6

図7　COLOR@に対する色の付け方

ち2つの画面で，図7のように共通になっています．この図に示されたことが，6箇所の円のすべてに成り立っているので，3画面分のデータが2画面分のデータだけで間に合っています．従って，48個（2×6×4）のデータだけで十分なのです．この様な工夫によって，ページングによる回転運動のアニメーションは，1個の円が回る所までこぎ付けたので，一応完成としましょう．

リスト7

```
100 '
110 '   demonstration of animation by paging
120 '
130  ON STOP GOSUB *EXIT : STOP ON
140  SCREEN 0,0 : CLS 3 : PRINT "Initialize !"
150  DIM CX1(2,5),CY1(2,5),CX2(2,5),CY2(2,5)
160 '
170  RESTORE *COLOR.DATA
180  FOR I=0 TO 5
190    READ CX1(0,I),CY1(0,I),CX2(0,I),CY2(0,I)
200    READ CX1(2,I),CY1(2,I),CX2(2,I),CY2(2,I)
210    CX1(1,I)=CX1(0,I) : CY1(1,I)=CY1(0,I)
220    CX2(1,I)=CX2(0,I) : CY2(1,I)=CY2(0,I)
230  NEXT
240 '
250  CLS : PRINT "Working !!"
260  FOR I=0 TO 2
270    COLOR 2^I : LOCATE 3,2
280    PRINT USING "BANK  ##";I+1
290    SCREEN 1,1,I,2^I  ←── 描き込み画面の選択
300    N=I*20
310    GOSUB *DRAW  ←── 6個の円を描くサブルーチン
320  NEXT
330 '
340  C=1
350 *LOOP
360  FOR J=0 TO 5
370    FOR I=0 TO 2
380      GOSUB *WAITR : BEEP 1 : BEEP 0
```

（190〜220行の注記：color@命令によって表示する範囲のデータを配列に読み込む）

```
390        COLOR 0 : CLS : SCREEN ,,,2^I    表示する画面(バンク)の決定
400        COLOR@(CX1(I,J),CY1(I,J))-(CX2(I,J),CY2(I,J)),C
410      NEXT                                  1つの円だけ表示する
420    NEXT
430    C=(C MOD 7)+1
440  GOTO *LOOP
450  '
460  *DRAW
470    WINDOW(-300,-180)-(300,180)
480    VIEW(0,0)-(639,199)
490    X=0 : Y=-100
500    SX=COS(2*3.1415/6)    : SY=SIN(2*3.1415/6)
510    SSX=COS(N*3.1415/90) : SSY=SIN(N*3.1415/90)
520  '
530    PP=X : X=SSX*X-SSY*Y }
540    Y=SSY*PP+SSX*Y        }
550  '                          { x=x·cosθ−y·sinθ
560    FOR I.D=1 TO 6          { y=x·sinθ+y·cosθ
570      P=X : X=SX*X-SY*Y }     の計算
580      Y=SY*P+SX*Y       }
590      CIRCLE(X,Y),15,7 : PAINT(X,Y),7
600    NEXT
610  RETURN
620  '
630  *WAITR
640    FOR W=0 TO 25 : NEXT
650  RETURN
660  '
670  *COLOR.DATA
680    DATA 33,4,42,7,29,6,33,8,25,8,30,12,25  }
690    DATA 12,29,14,26,14,33,18,33,17,37,19   } 1つの円をcolor命令で
700    DATA 38,18,46,20,46,16,50,18,49,12,55   } 表示させるためのデータ
710    DATA 16,51,10,55,12,46,6,53,10,42,4,46,7 }
720  '
730  *EXIT
740    COLOR 7 : CLS
750    SCREEN 0,0 : STOP OFF
760  RETURN *FIN
770  *FIN : END
```

7.4 GET@／PUT@の利用

さて，いよいよアニメーションの最後のテクニックの紹介となりました．アニメーションの1コマとなるパターンは，パレットを用いた場合7種類(黒を除く)，ページングを用いた場合3種類でしたが，今回の GET@／PUT@を用いたものは，やる気になれば，配列の許す限り多くの種類のコマ(パターン)を使うことができます．但し，GET@や PUT@には1つの欠点があり，割と大きなパターンやカラーのパターンの時，PUT@を用いると時間がかかるため，画面にパターンを描くのが見えてしまって，アニメーションらしくなりません．従って，再び白黒で小さなパターンのアニメーションをすることになります．

しかし，今度の場合は多くのパターンを作成できるので，パターンをコマごとに自由に描いていって，それを配列上に GET@文によって編集していき，PUT@文でアニメーションとして再生するようなツール――アニメーション・ジェネレータを作ってみましょう．これを解説することによって，GET@／PUT@によるアニメーションの手法を紹介したいと思います．

プログラムは，リスト8に示したもので，その構造は，ほぼ4章に挙げたグラフィック・エディタと同じです．異なる点は，アニメーションを白黒モードで利用するため，PAINT 文に関す

図8 アニメーション・ジェネレータの最初の表示

る機能が全くなくなり，色についての考え方も削除したことと，アニメーションを操作するコマンドが付いたことです．従って，色に関するコマンド以外は，すべてグラフィック・エディタと同じですから説明を省略します．また，クロス・カーソルを用いるため機械語のパッケージが必要となりますが，イニシャライズの方法，その他についても，4章のツールの所で解説済みですので，そちらの方を読んでおいてください．

リスト8において，アニメーションに関するサブルーチンは，ほとんど＊CLR．FRAMEより後にまとめてあり，それ以前では，イニシャライズの一部にアニメーションに関するものが含まれています．

プログラムを実行させた時に，アニメーションのコマに用いるフレームが，＊GR．INITで示されるサブルーチンの最後のFOR～NEXTループで，図8の様に描かれます．全部で，21個のフレームが出力され，それに左上から右向きに0～20の番号を振り，これをコマのページ数とします．そこで，それぞれのコマのフレームを，グラフィック画面のどこに描くのかを決定するために，ページ数を与えるとフレームの左上の座標の値を返してくる関数を，＊INITIALIZEの中のDEF FN文で定義しています．ここで定義されたFNCELLX(P)というのは，ページ数Pを与えると，フレーム左上の点のx座標の値を返し，FNCELLY(P)は，y座標の値を返してきます．＊GR．INITでフレームを描く時には，この2つの関数を用いており，＊CLR．FRAME以降のアニメーションに関するサブルーチンでも頻繁に使っています．では，早速リスト8に示した，アニメーション・ジェネレータの使い方について説明していきましょう．

パターンを描くコマンドは，グラフィック・エディタと全く同じです．但し，先程も述べた色に関するコマンドと画面消去，GET＠／PUT＠についての機能は持っていません．しかし，スペース・バーを押せば，緑色と黒の2色だけ交互に選ぶことができます．これは，黒を用いて画面の不要な部分を消すためです．

SHIFT ＋ N キーは，着目しているコマを次のコマに移します．画面上では，着目しているコマのフレームが緑色になっています．着目しているコマが移動する時，今，着目しているコマのフレームを白くするサブルーチンが＊CLR．FRAMEで，次に着目するコマのフレームを緑色にするサブルーチンが＊SET．FRAMEです．クロス・カーソルでパターンを描く所には，着目しているコマに何かパターンがあるとき，それが緑色で表示され，その1コマ前にパターンがあるとき，赤色でそのパターンが重なって表示されます．＊SET．FRAMEでは，それらの処理も

パターン作成

SHIFT ＋ G

着目ページの移動

SHIFT ＋ N, B, P

図9　アニメーション・ジェネレータの主な流れ

行っています．但し，赤いパターンは，上から重ねて描くと消えてしまいますが，この症状は N_{88}-BASICだけで解決しようとすると，パターンを描く処理に時間がかかるプログラムになってしまうので，あえてここではそのままにしてあります（機械語に頼れば楽に解決できます）．

SHIFT ＋ B キーは，SHIFT ＋ N キーと逆の操作で，着目しているコマを1つ前のコマに移します．SHIFT ＋ N，SHIFT ＋ B の両コマンド共，0ページと20ページが，あたかもループ状につながっている様になっており，0ページを着目している時に SHIFT ＋ B キーを押せば，着目ページは，20ページになります．もちろん，20ページに着目している時，SHIFT ＋ N キーを押せば0ページに着目ページが移ります．

しかし，以上2つのコマンドで，着目ページを変えるのは，場合によってはかなりキー操作の手間がかかります．そこで，直接着目しているコマのページ数を変えるために用意したコマンドが，SHIFT ＋ P キーです．このキーを押すと画面の各コマの上には，ページの番号が表示され，

Page number, please.

と表示されるので，目的のページ数をキーボードで入力してください．着目ページは，ここで入力したページに移ります．

作成したパターンは，SHIFT ＋ G キーにより，着目しているコマに格納されます．この際，着目ページは変化しません．要するに，アニメーション・ジェネレータの基本的な使い方は，図9に示したフロー・チャートの様になります．そして，ひと通り完成したアニメーションのパターンは，SHIFT ＋ A キーを押すことによって，クロス・カーソルでパターンを作成していた所に，再生されます．SHIFT ＋ A キーの処理をしているのが，＊ANIME.Aのサブルーチンですが，ここでは，PUT@にPSET のオプションを付けてあります．このオプションを用いると，前に PUT@したパターンが，次のパターンをPUT@することによって消えるので，アニメーションには非常に有効です．また，パターンの格納されていないページを PUT@しようとすると，普通 Illegal function call になるのですが，エラー・トラップによって，うまく避けるようにし

てあります．

さて，不用になったり，気に入らなくなったパターンを消去するコマンドも必要になってくるでしょう．そこで，用意したコマンドが SHIFT ＋ D シ キーです．このキーを押すと，現在着目しているコマとクロス・カーソルが表示されている部分のパターンが消去されます．もちろん，このコマンドで，配列の中もクリアすることは言うまでもないでしょう．

もう１つ用意した便利なコマンドが， SHIFT ＋ C ソ キーで，現在着目しているページのパターンを任意のページにコピーすることができます． SHIFT ＋ C ソ キーを押すと， SHIFT ＋ P セ キーを押した時の様に，ページの番号が表示され，

Copy page, please.

と，入力待ちになりますので，ページの番号をキーボードから入力してください．この処理をしているのが，＊COPY．Aです．

この様に作成したパターンも， SHIFT ＋ A チ で実行するだけでは，苦労の割に面白くありませんので，セーブとロードの機能と，配列上のパターンを画面に数値データとして表示する機能も持たせました．

セーブとロードは，Gエディタと同じで， SHIFT ＋ S ト ， SHIFT ＋ L リ キーで実行されます．着目しているページのパターンを数値データにするコマンドは，リスト８の＊TRANCE．Aで処理され， SHIFT ＋ T カ キーで動作します．データを書き写したら，スペース・バーを押せば，再び，アニメーション・ジェネレータが実行を継続します．

機能の概要は，以上の様なものですが，GET＠／PUT＠を用いたアニメーションの原理について少々説明を加えておきます．

GET＠文で取り込まれるパターンは，配列 CELL の中に，21コマ分のエリアが順番に並んでいます．ここに並んでいるパターンをただ単に画面に表示するだけでは，前に表示したパターンと重なってしまって，アニメーションになりません．前々から何度も説明している様に，アニメーションの基本的な流れは，7．1に挙げたアルゴリズム［Ｉ］のようなものです．今までの例(GET＠／PUT＠以外）では，［Ｉ］の②の消す作業を，本当の意味で〝消す〟ことによって行ってきました．ところが，PUT＠文の PSET オプションを用いると，既に描かれていた場所に，次のパターンを上から乗せる様に重ねて描くので，前のパターンは失なわれ，結局［Ｉ］の②と③の作業が一度にできたことになります．この機能が，もっと広い範囲に高速に利用できれば，かなり希望に沿ったアニメーションになることは間違いありません．

以上で PC-8801 (N_{88}-BASIC) を使ったアニメーションについての解説は終わりです．８ビットのマイクロコンピュータで処理される BASIC である以上，スピードやメモリとの関係で，N_{88}-BASIC を用いて，今テレビでやっているような滑らかなアニメーションは，無理と言い切れる

でしょう．ですが，工夫すればこの章で挙げた例ぐらいのアニメーションなら，可能なのですから，何かに挑戦してみてはいかがでしょうか．アニメーション・ジェネレータも，拡大や縮小，7．2のリスト5の様な機能などを付け加えれば，面白いものになると思いますので，是非改良してみてください．

リスト8

```
1000 '
1010 ' << Animetion Generator >> 注意）必ずGlibが必要
1020 '
1030  CLEAR ,&HE3FF          ┐
1040  GOSUB *INITIALIZE      ├ メインルーチン
1050  GOSUB *MAIN.LOOP       ┘
1060 END
1070 '
1080 ' << ｱﾆﾒｰｼｮﾝ ｼﾞｪﾈﾚｰﾀ ﾉ ｲﾆｼｬﾗｲｽﾞ >>
1090 '
1100 *INITIALIZE
1110  WIDTH 80,25 : CONSOLE 6,2 : CLS
1120 '
1130  DEFINT A-Z
1140  DEF FNCELLX(P)=(P MOD 7)*88
1150  DEF FNCELLY(P)=(P¥7)*48+64
1160  DEF USR=&HE400
1170 '
1180  ON ERROR GOTO *TRAP
1190  ON HELP GOSUB *MSG.BLOCK : HELP ON
1200 '
1210 *N88.SYSTEM
1220  COLOR 7 : SCREEN ,2 : CONSOLE 0,25 : CLS                   ┐
1230  LOCATE 25,7 : PRINT "Your N88-BASIC is ....."              │ 使用しているBASICが
1240  LOCATE 25,9 : PRINT "    1) Disk version"                  ├ Disk版かROM版かの
1250  LOCATE 25,10 : PRINT "    2) Rom version"                  │ チェック
1260  LOCATE 25,12 : INPUT "  number please.  ",BASIC            │
1270  IF BASIC<>1 AND BASIC<>2 THEN *N88.SYSTEM                  ┘
1280 '
1290  GOSUB *GR.INIT
1300 RETURN
1310 '
1320 ' << ｱﾆﾒｰｼｮﾝ ｼﾞｪﾈﾚｰﾀ ﾉ ﾒｲﾝﾙｰﾁﾝ >>'
1330 '
1340 *MAIN.LOOP
1350  SCREEN 0,0 : CONSOLE 6,2 : CLS
1360  WINDOW(DISPX,DISPY)-(XW.G,YW.G) : VIEW(DISPX,DISPY)-(XW.G,YW.G)
1370  GOSUB *LPSET.G
1380  GOSUB *INFORM.G
1390 '
1400  DUMMY=USR(GCREV.)←―― クロス・カーソルの表示
1410  COMMAND$=" srlbfcamSLABCDGNPT"←―― コマンド列の登録
1420  GOSUB *IN.KEY
1430 '
1440 *LP.A : IF CH$=CHR$(&H1B) THEN *EX.A  ESCキーなら＊EX.Aへ
1450    IF CH$="" THEN *CURSOR.G             ┐
1460    VEC.=INSTR(COMMAND$,CH$)             ├ コマンド解析
1470    IF VEC.=0 THEN *CURSOR.G             ┘
1480 '
1490    DUMMY=USR(GCREV.)←―― クロス・カーソルの消去           各処理の振り分け
1500    IF VEC.>11 THEN *A.COM                               ↙（グラフィック関係）
1510    ON VEC. GOSUB *COLOR.G,*PSET.G,*PRESET.G,*LINE.G,*LINEB.G,*LIN
EBF.G,*CIRCLE.G,*LPSET.G,*MODE.G,*SAVE.A,*LOAD.A
1520    DUMMY=USR(GCREV.) : GOTO *CURSOR.G←―― クロス・カーソルの表示
1530 '
1540   *A.COM                   ↙各処理の振り分け（アニメーション関係）
1550    ON VEC.-11 GOSUB *ANIME.A,*BACK.A,*COPY.A,*DELETE.A,*GET.A,*NE
XT.A,*PAGE.A,*TRANCE.A
1560    DUMMY=USR(GCREV.)←―― クロス・カーソルの表示
1570 '
1580   *CURSOR.G
1590    IF MODE.G=1 THEN PRESET(X,Y) ┐
1600    IF MODE.G=2 THEN PSET(X,Y)   ┘ モードによるクロス・カーソルの制御
```

```
1610     DUMMY=USR(GCMOV.)←―― クロス・カーソルの移動
1620     LOCATE 20,1 : PRINT USING "[ x=##  y=## ]";X,Y : LOCATE 0,7
1630 '                                 ↖着目座標の表示
1640     GOSUB *IN.KEY キー・コマンドの入力
1650 GOTO *LP.A
1660 '
1670 *EX.A : ON ERROR GOTO 0
1680  CONSOLE 0,25
1690 RETURN
1700 '
1710 ' << ｷｰ ﾉ ﾆｭｰﾘｮｸ >>
1720 '
1730 *IN.KEY
1740  CH$=INKEY$
1750 RETURN
1760 '
1770 ' << ｸﾞﾗﾌｨｯｸ ｶﾝｹｲ ﾉ ｲﾆｼｬﾗｲｽﾞ >>
1780 '
1790 *GR.INIT
1800  CLS : COLOR 7,0,0,4
1810  XW.G=69 : YW.G=34                                          ┐
1820  X=XW.G/2 : Y=YW.G/2                                        │
1830 '                                                           ├ 使用する変数の登録
1840  VEC.=0 : MODE.G=0 : R.G=0 : PAGE=0 : C.G=4                 │
1850  K.A=0 : L.A=0 : I.$="" : FILE$="" : MSG$=""                ┘
1860 '
1870  CLSM.=0 : CLSS.=1 : CLSH.=2 : GCREV.=14 : GCMOV.=15                 ┐
1880  W.=0 : WP.=VARPTR(W.) : P.=&HE5F8                                  │
1890  W.=VARPTR(X)     : POKE P.  ,PEEK(WP.) : POKE P.+1,PEEK(WP.+1)     │
1900  W.=VARPTR(Y)     : POKE P.+2,PEEK(WP.) : POKE P.+3,PEEK(WP.+1)     ├
1910  W.=VARPTR(XW.G) : POKE P.+4,PEEK(WP.) : POKE P.+5,PEEK(WP.+1)      │
1920  W.=VARPTR(YW.G) : POKE P.+6,PEEK(WP.) : POKE P.+7,PEEK(WP.+1)      ┘
1930  SCREEN 0,0 : DUMMY=USR(CLSM.)                  グラフィック・ライブラリに関する
1940 '                                               イニシャライズ
1950  BYTE.G=((XW.G+8)¥8)*(YW.G+1)+4
1960  FAC.G=BYTE.G¥2+1
1970  DIM CELL(FAC.G*21)
1980 '
1990  SCREEN 0,1
2000  FOR I.A=0 TO 21
2010    LINE(FNCELLX(I.A),FNCELLY(I.A))-STEP(XW.G,YW.G),7,B
2020  NEXT
2030  SCREEN 0,0
2040  DISPX=0 : DISPY=0
2050  GOSUB *SET.FRAME
2060 RETURN
2070 '
2080 ' << ｸﾞﾗﾌｨｯｸ ｶﾝｹｲ ﾉ ｻﾌﾞﾙｰﾁﾝ >>
2090 '
2100 *PSET.G             ┐
2110  PSET(X,Y)          ├ sキーの処理
2120  GOSUB *LPSET.G     ┘
2130 RETURN
2140 '
2150 *PRESET.G           ┐
2160  PRESET(X,Y)        ├ rキーの処理
2170  GOSUB *LPSET.G     ┘
2180 RETURN
2190 '
2200 *LINE.G                ┐
2210  LINE(LPX,LPY)-(X,Y)   ├ lキーの処理
2220  GOSUB *LPSET.G        ┘
2230 RETURN
2240 '
2250 *LINEB.G                    ┐
2260  LINE(LPX,LPY)-(X,Y),,B     ├ bキーの処理
2270  GOSUB *LPSET.G             ┘
2280 RETURN
2290 '
2300 *LINEBF.G                   ┐
2310  LINE(LPX,LPY)-(X,Y),,BF    ├ fキーの処理
2320  GOSUB *LPSET.G             ┘
2330 RETURN
```

```
2340 '
2350 *CIRCLE.G
2360  R.G=SQR((X-LPX)^2+4*(Y-LPY)^2)   } cキーの処理
2370  IF R.G=0 THEN RETURN
2380  CIRCLE(LPX,LPY),R.G
2390 RETURN
2400 '
2410 *LPSET.G                        } aキーの処理
2420  LPX=X : LPY=Y
2430 RETURN
2440 '
2450 *COLOR.G                        } クロス・カーソルのモードを変える
2460  C.G=C.G XOR 4 : COLOR ,,,C.G     mキーの処理
2470  GOSUB *INFORM.G
2480 RETURN
2490 '
2500 *MODE.G                         } スペース・キーの処理
2510  MODE.G=MODE.G+1                   (フォアグランド・カラーの処理)
2520  IF MODE.G>2 THEN MODE.G=0
2530  GOSUB *INFORM.G
2540 RETURN
2550 '
2560 *INFORM.G                       } アニメーション・
2570  LOCATE 20,1 : PRINT USING "[ x=##  y=## ]";X,Y    ジェネレータの
2580  LOCATE 20,2 : PRINT "Fore ground color = ";      各種情報
2590  IF C.G=4 THEN PRINT "green." ELSE PRINT "black."   の処理
2600  LOCATE 20,3 : PRINT "Mode is ";
2610  IF MODE.G=0 THEN PRINT "MOVE  ";
2620  IF MODE.G=1 THEN PRINT "PRESET ";
2630  IF MODE.G=2 THEN PRINT "PSET  ";
2640  PRINT "mode.   "
2650  LOCATE 0,7
2660 RETURN
2670 '
2680 ' << ｱﾆﾒｰｼｮﾝ ｼﾞｪﾈﾚｰﾀ ｶﾝｹｲ ﾉ ｻﾌﾞﾙｰﾁﾝ >>
2690 '
2700 *CLR.FRAME
2710  SCREEN 0,0              ↙白い枠を描く
2720  LINE(FNCELLX(PAGE),FNCELLY(PAGE))-STEP(XW.G,YW.G),7,B
2730  WINDOW(DISPX,DISPY)-(XW.G,YW.G) : VIEW(DISPX,DISPY)-(XW.G,YW.G)
2740 RETURN
2750 '
2760 *SET.FRAME
2770  SCREEN 0,0
2780  LINE(DISPX,DISPY)-STEP(XW.G,YW.G),0,BF ←── クロス・カーソルのある部分を消去
2790  SCREEN 1,,2,7
2800  PUT@(DISPX,DISPY),CELL(PAGE*FAC.G),PSET ←── 着目ページのセルを緑のバンクにPUT
2810  SCREEN ,,1,7
2820  IF PAGE=0 THEN I.A=20 ELSE I.A=PAGE-1
2830  PUT@(DISPX,DISPY),CELL(I.A*FAC.G),PSET ←─その前のページのセルを赤のバンクにPUT
2840  SCREEN 0,0
2850  LINE(FNCELLX(PAGE),FNCELLY(PAGE))-STEP(XW.G,YW.G),4,B ←── 緑の枠を描く
2860  WINDOW(DISPX,DISPY)-(XW.G,YW.G) : VIEW(DISPX,DISPY)-(XW.G,YW.G)
2870 RETURN
2880 '
2890 *PAGE.DISP                      } それぞれの枠に
2900  CONSOLE 0,25                     ページ番号を振る
2910  LOCATE 1,9
2920  PRINT "page 0      page 1      page 2      page 3";
2930  PRINT "      page 4      page 5      page 6"
2940  LOCATE 1,15
2950  PRINT "page 7      page 8      page 9      page 10";
2960  PRINT "     page 11     page 12     page 13"
2970  LOCATE 1,21
2980  PRINT "page 14     page 15     page 16     page 17";
2990  PRINT "     page 18     page 19     page 20"
3000 RETURN
3010 '
3020 ' << [  A  ] ｷｰ ﾉ ｼｮﾘ >>
3030 '
3040 *ANIME.A
```

```
3050  SCREEN 1,,1,2
3060  FOR I.A=0 TO 20
3070    PUT@(DISPX,DISPY),CELL(I.A*FAC.G),PSET
3080  NEXT
3090  GOSUB *SET.FRAME
3100 RETURN
3110 '
3120 ' << [  B  ] ｷｰ ﾉ ｼｮﾘ >>
3130 '
3140 *BACK.A
3150  GOSUB *CLR.FRAME
3160  IF PAGE=0 THEN PAGE=21
3170  PAGE=PAGE-1
3180  GOSUB *SET.FRAME
3190 RETURN
3200 '
3210 ' << [  C  ] ｷｰ ﾉ ｼｮﾘ >>
3220 '
3230 *COPY.A
3240  GOSUB *CLR.FRAME
3250  GOSUB *PAGE.DISP
3260  LOCATE 0,6
3270  INPUT "Copied page, please.",PAGE
3280  IF PAGE<0 AND PAGE>20 THEN PRINT "Not exist !" : GOTO *COPY.A
3290  CLS : CONSOLE 6,2 : CLS
3300  SCREEN 1,,2,4
3310  GET@(DISPX,DISPY)-STEP(XW.G,YW.G),CELL(PAGE*FAC.G)
3320  SCREEN 0,0
3330  PUT@(FNCELLX(PAGE),FNCELLY(PAGE)),CELL(PAGE*FAC.G),PSET,7,0
3340  GOSUB *INFORM.G
3350  GOSUB *SET.FRAME
3360 RETURN
3370 '
3380 ' << [  D  ] ｷｰ ﾉ ｼｮﾘ >>
3390 '
3400 *DELETE.A
3410  SCREEN 0,0
3420  LINE(FNCELLX(PAGE),FNCELLY(PAGE))-STEP(XW.G,YW.G),0,BF
3430  SCREEN 1,,2,7
3440  GET@(FNCELLX(PAGE),FNCELLY(PAGE))-STEP(XW.G,YW.G),CELL(PAGE*FAC.
G)
3450  GOSUB *SET.FRAME
3460 RETURN
3470 '
3480 ' << [  G  ] ｷｰ ﾉ ｼｮﾘ >>
3490 '
3500 *GET.A
3510  SCREEN 1,,2,7
3520  GET@(DISPX,DISPY)-STEP(XW.G,YW.G),CELL(PAGE*FAC.G)
3530  SCREEN 0,0
3540  PUT@(FNCELLX(PAGE),FNCELLY(PAGE)),CELL(PAGE*FAC.G),PSET,7,0
3550  GOSUB *SET.FRAME
3560 RETURN
3570 '
3580 ' << [  L  ] ｷｰ ﾉ ｼｮﾘ >>
3590 '
3600 *LOAD.A
3610  INPUT "Loaded file name, please : ",FILE$
3620  IF BASIC=2 THEN GOSUB *CAS.LOAD ELSE BLOAD FILE$,VARPTR(CELL(0))
3630  SCREEN 0,0
3640  FOR I.A=0 TO 20
3650    PUT@(FNCELLX(I.A),FNCELLY(I.A)),CELL(I.A* FAC.G),PSET,7,0
3660    LINE(FNCELLX(I.A),FNCELLY(I.A))-STEP(XW.G,YW.G),7,B
3670  NEXT
3680  PRINT : PRINT USING ">> @ is loaded <<";FILE$
3690  GOSUB *SET.FRAME
3700 RETURN
3710 '
3720 *CAS.LOAD
3730  CONSOLE 0,25 : CLS : SCREEN ,2
3740  PRINT USING "Loading from  @  to  CELL";FILE$
3750  MON : CLS : CONSOLE 6,2 : CLS
3760  GOSUB *INFORM.G
```

注釈:
- 3050〜3100: ページ順にセルを表示 これがアニメーションを実行するサブルーチン
- 3140〜3190: 1ページ前のセルに着目する
- 3270: 着目ページを任意のページにコピー
- 3420: 着目ページのセルを消す
- 3440: 消去したセルをGETする
- 3520: クロス・カーソルのあるセルをGET
- 3620: ディスク版ならパターンをロード
- 3650〜3670: ロードしたパターンを表示
- 3730〜3760: モニタを用いて，カセット・テープからロード

```
3770 RETURN
3780 '
3790 ' << [  N  ] ｷｰ ﾉ ｼｮﾘ >>
3800 '
3810 *NEXT.A
3820  GOSUB *CLR.FRAME
3830  IF PAGE=20 THEN PAGE=-1
3840  PAGE=PAGE+1
3850  GOSUB *SET.FRAME
3860 RETURN
3870 '
3880 ' << [  P  ] ｷｰ ﾉ ｼｮﾘ >>
3890 '
3900 *PAGE.A←――――― 1ページ先のセルに着目する
3910  GOSUB *CLR.FRAME
3920  GOSUB *PAGE.DISP
3930  LOCATE 0,6
3940  INPUT "Page number, please. ",PAGE
3950  IF PAGE<0 AND PAGE>20 THEN PRINT "Not exist !" : GOTO *PAGE.A
3960  CLS : CONSOLE 6,2 : CLS
3970  GOSUB *INFORM.G
3980  GOSUB *SET.FRAME
3990 RETURN
4000 '
4010 ' << [  S  ] ｷｰ ﾉ ｼｮﾘ >>
4020 '
4030 *SAVE.A
4040  GOSUB *PAGE.DISP
4050  LOCATE 0,6
4060  INPUT "Upper page for save : ",I.A
4070  BYTE.G=(((XW.G+8)¥8)*(YW.G+1)+4)*(I.A+1)
4080  INPUT "Saved file name, please : ";FILE$   ↙ディスクへパターンをセーブ
4090  IF BASIC=2 THEN GOSUB *CAS.SAVE ELSE BSAVE FILE$,VARPTR(CELL(0))
,BYTE.G
4100  PRINT : PRINT USING ">> @ is saved <<";FILE$
4110  CONSOLE 6,2 : GOSUB *INFORM.G
4120  GOSUB *SET.FRAME
4130 RETURN
4140 '
4150 *CAS.SAVE←――――― モニタを用いて，カセット・テープへセーブ
4160  CLS
4170  SCREEN ,2 : I.A=VARPTR(CELL(0))
4180  PRINT USING "Saving from  CELL  to  @";FILE$
4190  PRINT USING "address from _&H@ to _&H@";HEX$(I.A),HEX$(I.A+BYTE.
G)
4200  MON : CLS : CONSOLE 6,2 : CLS : SCREEN 0,0
4210 RETURN
4220 '
4230 ' << [  T  ] ｷｰ ﾉ ｼｮﾘ >>
4240 '
4250 *TRANCE.A
4260  I.A=PAGE                                         ┐
4270  SCREEN ,2 : CONSOLE 0,25 : CLS                    │
4280  FOR K.A=I.A*FAC.G TO ((I.A+1)*FAC.G-1) STEP 10   │ 着目ページを
4290    FOR L.A=0 TO 9                                  │ 数値に変換して表示
4300      PRINT STR$(CELL(K.A+L.A));                    │
4310      IF L.A<>9 THEN PRINT ",";                     ┘
4320    NEXT
4330    PRINT
4340  NEXT
4350  PRINT : PRINT "ﾃﾞｰﾀ ｦ ｳﾂｼ ﾀﾗ ﾅﾆｶ ｷｰ ｦ ｵｼﾃｸﾀﾞｻｲ｡"
4360 *WAIT.LOOP.A : IF INKEY$="" THEN *WAIT.LOOP.A
4370  CLS : CONSOLE 6,2 : CLS
4380  GOSUB *INFORM.G
4390  GOSUB *SET.FRAME
4400 RETURN
4410 '
4420 ' << ｱﾆﾒｰｼｮﾝ ｼﾞｪﾈﾚｰﾀ ﾉ [HELP] ﾒｯｾｰｼﾞ >>
4430 '
4440 *MSG.BLOCK
4450  HELP OFF : SCREEN ,2 : CONSOLE 0,25
4460  CLS : PRINT TAB(8);"*** Animation Generator Command Summary ***"
4470  PRINT : PRINT TAB(40);"(graphic & cross-cursor control)"
```

```
4480  LOCATE 0,5
4490  PRINT "[space]   :    フォア・ク゛ラウント゛・カラー ノ コート゛ハ゛ンコ゛ウ ラ カエマス。"
4500  PRINT "[  s  ]   :    チャクモクサ゛ヒョウ (x,y) ニ テン ラ ウチマス。  ( PSET )"
4510  PRINT "               LP ハ コノ サ゛ヒョウ (x,y) ニ ナリマス。"
4520  PRINT "[  r  ]   :    チャクモクサ゛ヒョウ (x,y) ノ テン ラ ケシマス。  ( PRESET )"
4530  PRINT "               LP ハ コノ サ゛ヒョウ (x,y) ニ ナリマス。"
4540  PRINT "[  l  ]   :    サ゛ヒョウ (lpx,lpy) カラ サ゛ヒョウ (x,y) マテ゛ チョクセン ラ
エカ゛キマス。  ( LINE )"
4550  PRINT "               LP ハ サ゛ヒョウ (x,y) ニ ナリマス。"
4560  PRINT "[  b  ]   :    'b' オフ゜ション ノ LINEメイレイ ニ ソウトウシマス。"
4570  PRINT "               LP ハ サ゛ヒョウ (x,y) ニ ナリマス。"
4580  PRINT "[  f  ]   :    'bf' オフ゜ション ノ LINEメイレイ ニ ソウトウシマス。"
4590  PRINT "               LP ハ サ゛ヒョウ (x,y) ニ ナリマス。"
4600  PRINT "[  c  ]   :    LP サ゛ヒョウ (lpx,lpy) ラ チュウシン ト スル エン ラ エカ゛キマス
。"
4610  PRINT "               ソシテ ハンケイ ハ SQR((x-lpx)^2+4*(y-lpy)^2) テ゛ アラ
ワサレマス。"
4620  PRINT "[  a  ]   :    LP ラ サ゛ヒョウ (lpx,lpy) カラ サ゛ヒョウ (x,y) ニ ウツシ、
セット シマス。"
4630  PRINT "[  m  ]   :    アニメーション・シ゛ェネレータ ノ モート゛ ラ カエマス。"
4640  LOCATE 0,22
4650  PRINT "[ ESC ]   :    Return to Animation Generator."
4660  PRINT "[ ^n  ]   :    ----------------------------------------
--<< another page >>."
4670 '
4680 *WAIT.LOOP.MB : I.$=INKEY$
4690  IF I.$=CHR$(&H1B) THEN *EXIT.MB
4700  IF I.$=CHR$(&HE) THEN *MORE.MB ELSE *WAIT.LOOP.MB
4710 '
4720 *MORE.MB
4730  CLS : PRINT TAB(8);"*** Animation Generator Command Summary ***"
4740  PRINT : PRINT TAB(40);"(animation generator & I/O control)"
4750  LOCATE 0,6
4760  PRINT TAB(16);"[  A  ]   :   アニメーション ラ シ゛ッコウ シマス。"
4770  PRINT TAB(16);"[  B  ]   :   チャクモク ヘ゜ーシ゛ ラ マエ ノ ヘ゜ーシ゛ ニ シマス。"
4780  PRINT TAB(16);"[  C  ]   :   チャクモク ヘ゜ーシ゛ ラ ヘ゛ツ ノ ヘ゜ーシ゛ ニ コヒ゜ー シマ
ス。"
4790  PRINT TAB(16);"[  D  ]   :   チャクモク ヘ゜ーシ゛ ラ ショウキョ シマス。"
4800  PRINT TAB(16);"[  G  ]   :   チャクモク ヘ゜ーシ゛ ニ ハ゜ターン ラ GET@ テ゛ ヒロイ マ
ス。"
4810  PRINT TAB(16);"[  L  ]   :   テ゛ィスケット カラ ハ゜ターン ラ ロート゛ シマス。"
4820  PRINT TAB(16);"[  N  ]   :   チャクモク ヘ゜ーシ゛ ラ ツキ゛ ノ ヘ゜ーシ゛ ニ シマス。"
4830  PRINT TAB(16);"[  P  ]   :   チャクモク ヘ゜ーシ゛ ラ ヘ゛ツ ノ ヘ゜ーシ゛ ニ ヘンコウ シマ
ス。"
4840  PRINT TAB(16);"[  S  ]   :   ハ゜ターン ラ テ゛ィスケット ニ セーフ゛ シマス。"
4850  PRINT TAB(16);"[  T  ]   :   チャクモク ヘ゜ーシ゛ ノ ハ゜ターン ラ テ゛ータ ニ シマス。"
4860  LOCATE 0,22
4870  PRINT "[ ESC ]   :    Return to Animation Generator."
4880  PRINT "[ ^n  ]   :    ----------------------------------------
--<< another page >>."
4890 *WAIT.LOOP.MB1 : I.$=INKEY$
4900  IF I.$=CHR$(&H1B) THEN *EXIT.MB
4910  IF I.$=CHR$(&HE) THEN *MSG.BLOCK ELSE *WAIT.LOOP.MB1
4920 '
4930 *EXIT.MB
4940  CLS : CONSOLE 6,2 : CLS
4950  SCREEN ,0
4960  GOSUB *INFORM.G : HELP ON
4970 RETURN
4980 '
4990 ' << エラー トラッフ゜ >>
5000 '
5010 *TRAP
5020  IF ERR=5 THEN RESUME NEXT
5030  COLOR 2
5040 '
5050  IF ERR=52 THEN PRINT "Bad file number"
5060  IF ERR=53 THEN PRINT "File not found"
5070  IF ERR=56 THEN PRINT "Bad file name"
5080  IF ERR=61 THEN PRINT "File write protect"
5090  IF ERR=62 THEN PRINT "Disk offline"
5100  IF ERR=64 THEN PRINT "Disk I/O error"
5110  IF ERR=68 THEN PRINT "Disk full"
```

```
5120  IF ERR=69 THEN PRINT "Bad allocation table"
5130  IF ERR=70 THEN PRINT "Bad drive number"
5140  IF ERR=72 THEN PRINT "Deleted record"
5150 '
5160  BEEP : COLOR 7 : RESUME *TRAP.1
5170 '
5180 *TRAP.1
5190  WINDOW(0,0)-(XW.G,YW.G) : VIEW(0,0)-(XW.G,YW.G)
5200 RETURN
```

8章 3次元グラフィックスへの招待

これまで座標系についてスクリーン座標系，ワールド座標系などいろいろな座標系の名前が出てきましたが，これらは N_{88}-BASIC のグラフィックスに固有な座標系で，他の BASIC にはありません．また今まで扱ってきたのは画面上の位置を示すための x 軸と y 軸の 2 つを持った座標系で，表示する図形も平面の図形でした．しかし，私達が住んでいるのは 3 次元の世界です．

そこで，この章ではこれまで述べてきた，2 次元グラフィックスでの図形の移動や回転などの 2 次元変換と，さらに 3 次元グラフィックスについて，一般的な理論を中心にお話ししましょう．

8.1 3次元グラフィックスとは

2 次元，3 次元など次元 (dimension) という言葉を聞く事があるかと思います．dimension とは DIM 文の DIM とも関係があり，難しくいうと「ある集合の中の元の位置を単独に，または他の量と関連させて規定する量」という事ですが，ここでは簡単に「位置を表す座標軸がいくつあるか」と考えてください．ちなみに，3 次元グラフィックスのことを 3 Dグラフィックスというのは，このためです．

(a) 1 次元

(b) 2 次元

(c) 3 次元

図 1　各次元における位置の表し方

1次元とは座標軸が1つ，つまり直線上の位置しか持たない世界です．2次元とは座標軸が2つ——普通x軸とy軸で表したりしますが——これが平面の世界です．画面は2次元の平面ですから，その上に表示されるのは当然のことながら2次元の図形です．

次に，3次元の世界について考えてみますが，これは2次元に，さらにもう1つの座標軸，この図ではz軸が加っただけの事です．私たちの住んでいる世界もこの3次元の世界で，物を立体として考えます．球を円と混同したりしませんし，遠近の区別もあります．しかし，この3次元の物体をグラフィックに表示するにはどうしたら良いでしょう．立体テレビや飛び出すグラフィック・ディスプレイと言ったものもありませんし，レーザ光線によるホログラフもまだ一般の用途に耐えるものではありません．3次元の物体をそのまま立体感を持たせて表示させる手段を，私達はまだ完全には持っていないのです．では，ここで説明する3次元グラフィックスとは何でしょう？

写真や絵を見て薄っぺらい平面の世界に見えますか？時には立体感のある迫力のある絵や写真を見た事もあるはずです．

物体に斜めから光を当てると影が出きます．前に物体があると，後の物体は隠れて見えなくなります．また，遠くの物体は小さく見えます．これらはあたり前の事と思うかもしれませんが，実は重要な事なのです．

紙に円を描いても円，あたり前ですね．しかし，これに適当な影を書き加えると球に見えてきます．つまり平面にでも立体をそれらしく描く事ができるのです（図2）．

(a) 遠近法の図

円　球

(b) 影による立体感

(c) 陰線の消去

図2　平面における3次元物体の表し方

さて平面にも立体図形を描く事ができるのならば，これを写真や絵だけではなく，計算機を使っても可能ではないかと考える様になりました．と言うのは，写真は，あるがままをフィルムに焼き付けるだけですし，絵は人手による作業です．ところが計算機は，プログラムとデータさえ与えれば，同じ事を何度でも行えますし，また物体の動きが，数式などプログラムできる形で与えられていれば，実際に物を動かさなくとも，計算により表示させて目で見る事ができるからで

す．最近の計算機や表示装置の向上により，これらの事は十分実現可能になり，SF映画やアニメーション，そして何げなく見ているテレビのCMにもコンピュータによる3次元グラフィックスが使われ始めています．

そこで，パーソナル・コンピュータでも3次元グラフィックスをやってみようというのがこの章の内容です．残念ながら，本格的な3次元グラフィックスを処理している計算機とパソコンとでは，その能力に大きな差があります．それで，ここでは3次元グラフィックスの基礎ぐらいしかお話しできませんが，大規模な3次元グラフィックスにも通じる所も多いので，物は試しと挑戦してみてください．

8.2 図形の2次元変換

3次元グラフィックスの前に，2次元グラフィックスを説明しましょう．3次元と言えども，表示するのは平面であるCRTですから，まずは2次元の世界です．

この節の題名に「2次元変換」とありますが，これは簡単に言えば，図形の移動，拡大・縮小，回転などの事です．

8.2.1 平行移動

ある図形を表示してみたが，もう少し右に表示し直したい場合どうしますか？ LINE 文でもCIRCLE文でも，その時のx方向の位置を与えている値を増すだけですね．実はこれだけでもう平行移動なのです．

平面上での平行移動は実に簡単で，次の式で与えられます．

$$\begin{cases} X=x+T_x \\ Y=y+T_y \end{cases}$$

この式により点(x, y)をx方向にT_x．y方向にT_y平行移動した場合，移動後の座標は(X, Y)となります(図3)．これで点AがA'へ移動します．このことは，図形が複雑になっても同じ事で，すべての点を平行移動させて，再び描けば良いのです．

図3　点の平行移動

さて，ここでベクトルと行列という考え方を導入します．多少難しいかもしれませんが，この考え方を用いると平行移動や後に出てくる回転，拡大・縮小も簡単に式として表すことができますし，3次元グラフィックスの基礎を知る上で重要です．

ここで説明した平行移動の式は，次の行列演算の式により導けます．

$$[X \quad Y \quad 1] = [x \quad y \quad 1] \begin{bmatrix} 1 & 0 & 0 \\ 0 & 1 & 0 \\ T_x & T_y & 1 \end{bmatrix}$$

この式を展開して求まるのが，前に述べた平行移動の式ですが，ここでは行列演算の何たるかについては述べませんので御存知ない方は，それなりの参考文献で勉強してください．

では，プログラミングの例として四角形の平行移動を行ってみましょう．

リスト1

```
1000 '
1010 ' 2D graphic  sample program
1020 '
1030 ' MOVE
1040 '
1050  SCREEN 0,3 : CLS 3 : SCREEN 0,0
1060  N=4 : DIM P(N,1)
1070  FOR I=0 TO N : READ P(I,0),P(I,1) : NEXT
1080  COL=2 : GOSUB *DISPLAY
1090  FOR J=1 TO 4
1100   TX=30 : TY= 6 : GOSUB *MOVE.2D
1110   COL=4 : GOSUB *DISPLAY
1120  NEXT
1130  END
1140 '
1150 DATA 90,30 ,160,140 ,350,130 ,290,40 ,90,30
1160 '
1170 *DISPLAY
1180  POINT (P(0,0),P(0,1))
1190  FOR I=1 TO N : LINE -(P(I,0),P(I,1)),COL : NEXT
1200  RETURN
1210 '
1220 *MOVE.2D
1230  FOR I=0 TO N
1240   P(I,0)=P(I,0)+TX : P(I,1)=P(I,1)+TY
1250  NEXT
1260  RETURN
```

8.2.2 拡大・縮少

拡大･縮小（スケーリングとも言いますが），これも平行移動と同じぐらい簡単な操作で行う事ができます．

ある図形を表示した時，その半分の大きさを描く時には各点のx，yの値を半分にすれば良いのです．つまり，

$$\begin{cases} X = x * S_x \\ Y = y * S_y \end{cases}$$

この2つの式を図形の各点について計算すると，元の図形のx方向に S_x 倍，y方向に S_y 倍の図形が得られます（図4）．S_x，S_y 共に1ならば，1倍の図形，つまり元の図形と同じになります．S_y を1とすると，x方向だけに S_x倍された図形が得られる訳です．また，S_x，S_y 共に−1とすれば原点に対称な図形になります．

ここで，S_x，S_y は0にすると大変です．x方向に0倍，y方向に0倍の図形，つまり点になってしまいます．

このスケーリングも，平行移動の時と同じ様な行列により表すことができます．

$$[X \quad Y \quad 1] = [x \quad y \quad 1] \begin{bmatrix} S_x & 0 & 0 \\ 0 & S_y & 0 \\ 0 & 0 & 1 \end{bmatrix}$$

次のプログラムで四角形の拡大･縮小を行いますが，その中心を四角形の内部とするために，拡大・縮小を行う前後で平行移動をしています．

図4　図形の縮少（拡大）

リスト2

```
1000 '
1010 ' 2D graphic  sample program
1020 '
1030 ' SCALING
1040 '
1050  SCREEN 0,3 : CLS 3 : SCREEN 0,0
1060  N=4 : DIM P(N,1)
1070  FOR I=0 TO N : READ P(I,0),P(I,1) : NEXT
1080  COL=2 : GOSUB *DISPLAY
1090  FOR J=1 TO 4
1100   TX=-160 : TY=-60 : GOSUB *MOVE.2D
1110   SX=1.2 : SY=1.1 : GOSUB *SCALE.2D
1120   TX= 160 : TY= 60 : GOSUB *MOVE.2D
1130   COL=4 : GOSUB *DISPLAY
1140  NEXT
1150  END
1160 '
1170 DATA 90,30 ,160,140 ,350,130 ,290,40 ,90,30
1180 '
1190 *DISPLAY
1200  POINT (P(0,0),P(0,1))
1210  FOR I=1 TO N : LINE -(P(I,0),P(I,1)),COL : NEXT
1220  RETURN
1230 '
1240 *MOVE.2D
1250  FOR I=0 TO N
1260   P(I,0)=P(I,0)+TX : P(I,1)=P(I,1)+TY
1270  NEXT
1280  RETURN
1290 '
1300 *SCALE.2D
1310  FOR I=0 TO N
1320   P(I,0)=P(I,0)*SX : P(I,1)=P(I,1)*SY
1330  NEXT
1340  RETURN
```

8.2.3 回転

さて図形の回転は，平行移動や拡大・縮少より少し難しくなりますが，次の式により表されます．実は，この式は2章の〝回転による円の描き方″と7章の〝ページングによるアニメーション″の所で紹介済みですから，覚えている方もおられるでしょう．

$$\begin{cases} X = \cos\theta * x + \sin\theta * y \\ Y = -\sin\theta * x + \cos\theta * y \end{cases}$$

この式により点（x，y）から原点を中心に時計回りに角度θ分移動した点（X，Y）が求まり

ます．また，この回転を行う行列式は，

$$[X \quad Y \quad 1] = [x \quad y \quad 1] \begin{bmatrix} \cos\theta & -\sin\theta & 0 \\ \sin\theta & \cos\theta & 0 \\ 0 & 0 & 1 \end{bmatrix}$$

で与えられます，但し，N_{88}-BASICのグラフィック画面上では，y軸が下向きに伸びている事を忘れないでください（図5）．

(a) 一般の2次元座標でのX軸，y軸　(b) N_{88}-BASICのスクリーン座標でのx軸，y軸

図5　一般的座標とN_{88}-BASICの座標の違い

次のプログラムでは円を，原点（0，0）を中心に回転させて表示していますが，回転の方向は反時計回りになっています(図6)．これも，ひとえにy軸が逆向きなためで，グラフィック画面上で，この演算により求まった点を表示する時はよく注意してください．

(a) 一般的な2次元座標での回転方向　(b) N_{88}-BASICのスクリーン座標での回転方向

図6

リスト3

```
1000 '
1010 ' 2D graphic  sample program
1020 '
1030 ' ROTATION
1040 '
1050  SCREEN 0,3 : CLS 2 : SCREEN 0,0 : WIDTH 80,25
1060  WINDOW (-100,-100)-(100,100) : VIEW (100,0)-(500,199)
1070  LINE (0,-100)-(0,100),1 : LINE (-100,0)-(100,0),1
1080  LOCATE 38,24 : PRINT "x";
1090  LOCATE 38,12 : PRINT "o";
1100  LOCATE 64,12 : PRINT "y";
1110  X=5 : Y= 90
1120  CIRCLE (X,Y),3,1
1130  ROL=7
1140 '
1150  RAD=3.1416*ROL/180
1160  FOR I=1 TO 50
1170   W=X*COS(RAD)+Y*SIN(RAD)
1180   Y=-X*SIN(RAD)+Y*COS(RAD)
1190   X=W
1200   CIRCLE (X,Y),3,4
1210  NEXT
1220 END
```

8.2.4 クリッピングとウィンドウ

ここまで，2次元での図形処理について，平行移動，回転，拡大・縮小と説明してきましたが，2次元図形に対する操作として，もう1つ，表示の時に行うクリッピングとウィンドウニングがあります．

クリッピングとは始めて聞く言葉のようですが，実は今までに N_{88}-BASICのグラフィックスで何度も使っていることで，WINDOW文とVIEW文がこれに関したステートメントなのです．

次のプログラムを実行すると，まず青で斜めに線を引き，次にウィンドウとビューポートを設定し，同じ様にLINE文を実行しますが，今度はビューポートの枠の内部だけに赤で描きます．

リスト4

```
100 '
110 ' CLIPING by WINDOW and VIEW
120 '
130 SCREEN 0,3 : CLS 3 : SCREEN ,0
140 FOR X=0 TO 320 STEP 20
150  LINE (X,0)-(300+X,199),1
160  LINE (300+X,0)-(X,199),1
170 NEXT
180 '
190 WINDOW (250,50)-(370,150)
200 VIEW (250,50)-(370,150),0,4
210 '
220 FOR X=0 TO 320 STEP 20
230  LINE (X,0)-(300+X,199),2
240  LINE (300+X,0)-(X,199),2
250 NEXT
260 END
```

この様にビューポート内だけを描画する様にクリップする事（摑み取る）がクリッピングです．

N_{88}-BASICの場合では，WINDOW文により，ワールド座標内に表示ウィンドウを切る事が，ウィンドウニングであり，VIEW文で画面上での表示領域を設定するのがビューイングなのです．

N_{88}-BASICには，WINDOW文，VIEW文という強力なステートメントがあるので，クリッピング等を特に意識せずに，グラフィック表示が行える訳です．但し，他のパーソナル・コンピュータのグラフィックで，この様な機能があるとは限りません．そこで，この N_{88}-BASIC特有の命令を使わずにクリッピング，ウィンドウニング，そしてビューイングを実現してみましょう．このことは3次元グラフィックスの基礎として非常に重要です．

☆**クリッピング**

クリッピングを行うには，いくつかの方法がありますが，ここでは直線を引く時の，除算を使ったクリッピングについて説明しましょう．

点 P_1 と点 P_2 を結ぶ直線を，図7に示す様な，点（CX_1，CY_1）と点（CX_2，CY_2）を対角の頂点とする長方形内だけに表示する様にクリッピングします．

まず始めに，直線の両端の点 P_1 と P_2 のそれぞれについて，それらが表示領域内にあるかどうか調べます．(a) の場合には2点共に領域内ですから，何も手を加えず引けばいいのです．次に (b) の場合はどうでしょう．一方の点だけが表示領域内です．そこで，この時には点 P_1，P_2 からなる直線上で，$x=CX_2$ なる点 P_3 を求めればいいのはすぐに分かると思います．

さて，問題なのは (c) や (d) の場合です．どちらの点も領域外にありますが，これをクリッピングした場合，(c) では点 P_3，P_4 からなる直線を引き，(d) では何も表示しないのです．この違いはどの様な条件を満たさなくて領域外にあるかを調べれば分かります．

(d) の場合には2点共に $x \leqq CX_2$ を満足していません．(c) では点 P_1 が $y \geqq CY_1$，点 P_2 が $x \leqq CX_2$ を満たしていません．つまり2点共に同じ条件を満足していない時にだけ，クリッピング後直線を引かないのです．そこで (c) の場合には，点 P_1，P_2 からなる直線上で $x=CX_2$ なる点P_3 を求め，再びこの P_1P_3 からなる直線についてクリッピングを繰り返す訳です．また (e) の場合も，(c) と同じ事をする訳ですが，その時求めた点 P_3 と P_1 を調べると，2点共に，$y \leqq CY_2$ を満していませんので，この場合には(d)と同じでクリッピング後は直線を引かないのです．

この様な考え方で実現したのが，次のクリッピング・プログラムで，＊CHECKが表示領域内にあるかどうかを調べるサブルーチン，そして＊CLIP以降がクリッピングのメインルーチンです．このルーチンを使うことで，BASICのWINDOWとVIEWを使ったプログラム（リスト4）と同じ事を実現しています．

リスト5

```
100 '
110 ' CLIPING by Program
120 '
130 SCREEN 0,3 : CLS 3 : SCREEN ,0
140 FOR I=0 TO 320 STEP 20
150  LINE (I,0)-(300+I,199),1
160  LINE (300+I,0)-(I,199),1
170 NEXT
180 '
190 CX1=250 : CX2=370 : CY1=50 : CY2=150 クリッピング領域の設定
200 LINE (CX1-1,CY1-1)-(CX2+1,CY2+1),0,BF
```

```
210 LINE (CX1-1,CY1-1)-(CX2+1,CY2+1),4,B
220 '
230 FOR I=0 TO 320 STEP 20
240  X1=I : X2=300+I : Y1=0 : Y2=199
250  GOSUB *CLIP
260  X1=300+I : X2=I : Y1=0 : Y2=199
270  GOSUB *CLIP
280 NEXT
290 END
300 '
1000 *CLIP クリッピングのサブルーチン
1010  X=X1 : Y=Y1 : GOSUB *CHECK : C1%=CVAL% } 直線の両端の点について,
1020  X=X2 : Y=Y2 : GOSUB *CHECK : C2%=CVAL% } ビューポート内かどうかのチェック
1030 *CLIP1
1040  IF C1%=0 AND C2%=0 THEN LINE (X1,Y1)-(X2,Y2),2 : RETURN ←両点ともに
                                                              領域内
1050  IF C1% AND C2% THEN RETURN ←同じ条件を満たさずに領域外
1060  IF C1% THEN C%=C1% ELSE C%=C2%          ビューポートの枠との交点を求める
1070  IF C% AND 1 THEN Y=Y1+(Y2-Y1)*(CX1-X1)/(X2-X1):X=CX1:GOTO *CLIP2
1080  IF C% AND 2 THEN Y=Y1+(Y2-Y1)*(CX2-X1)/(X2-X1):X=CX2:GOTO *CLIP2
1090  IF C% AND 4 THEN X=X1+(X2-X1)*(CY1-Y1)/(Y2-Y1):Y=CY1:GOTO *CLIP2
1100  IF C% AND 8 THEN X=X1+(X2-X1)*(CY2-Y1)/(Y2-Y1):Y=CY2
1110 *CLIP2
1120  IF C%=C1% THEN X1=X:Y1=Y:GOSUB *CHECK:C1%=CVAL% ELSE X2=X:Y2=Y:G
OSUB *CHECK:C2%=CVAL%
                    ↖ 求めた点について再び領域内かどうかをチェック
1130  GOTO *CLIP1
1140 '
1150 *CHECK  領域内かどうかをチェックするサブルーチン
1160  CVAL%=0
1170  IF X<CX1 THEN CVAL%=1
1180  IF CX2<X THEN CVAL%=CVAL%+2
1190  IF Y<CY1 THEN CVAL%=CVAL%+4
1200  IF CY2<Y THEN CVAL%=CVAL%+8
1210  RETURN
```

a)　b)　c)　d)　e)

図7　直線のクリッピング

☆ビューとウィンドウ

クリッピングさえできれば，ビューイングやウィンドウニングは簡単に行えます．

図の様にウィンドウを設定し，それを画面上のビューポート内に表示するには，まず次の2式を使います．

$$X=\frac{VX_2-VX_1}{WX_2-WX_1}(x-WX_1)+VX_1$$

$$Y=\frac{VY_2-VY_1}{WY_2-WY_1}(y-WY_1)+VY_1$$

この式によりx，yはスクリーン上での位置X，Yに変換されます．そしてこのXとYを点(VX_1, VY_1）と点（VX_2，VY_2）を頂点とした長方形内のみに表示する様にクリッピングを行うと，表示はビューポート内だけになります．

図8　ウィンドウとビューポート

リスト6

```
100 '
110 ' view and window
120 '
130 SCREEN 0,3 : CLS 3 : SCREEN ,0
140 FOR I=0 TO 320 STEP 40
150  LINE (I,0)-(300+I,199),1
160  LINE (300+I,0)-(I,199),1
170 NEXT
180 '
190 WX1=0 : WX2=639 : WY1=0 : WY2=199
200 VX1=300 : VX2=516 : VY1=100 : VY2=166
210 LINE (VX1-1,VY1-1)-(VX2+1,VY2+1),4,B
220 LINE (VX1,VY1)-(VX2,VY2),0,BF
230 '
240 FOR I=0 TO 320 STEP 40
250  X1=I : X2=300+I : Y1=0 : Y2=199
260  GOSUB *WINVIEW
270  X1=300+I : X2=I : Y1=0 : Y2=199
280  GOSUB *WINVIEW
290 NEXT
300 END
310 '
```

```
320 *WINVIEW
330  X1=(VX2-VX1)*(X1-WX1)/(WX2-WX1)+VX1 ┐ウィンドウからビューポートへの
340  Y1=(VY2-VY1)*(Y1-WY1)/(WY2-WY1)+VY1 │座標変換ルーチン
350  X2=(VX2-VX1)*(X2-WX1)/(WX2-WX1)+VX1 │
360  Y2=(VY2-VY1)*(Y2-WY1)/(WY2-WY1)+VY1 ┘
370 '
380 *CLIP
390  X=X1 : Y=Y1 : GOSUB *CHECK : C1%=CVAL%
400  X=X2 : Y=Y2 : GOSUB *CHECK : C2%=CVAL%
410 *CLIP1
420  IF C1%=0 AND C2%=0 THEN LINE (X1,Y1)-(X2,Y2),2 : RETURN
430  IF C1% AND C2% THEN RETURN
440  IF C1% THEN C%=C1% ELSE C%=C2%
450  IF C% AND 1 THEN Y=Y1+(Y2-Y1)*(VX1-X1)/(X2-X1):X=VX1:GOTO *CLIP2
460  IF C% AND 2 THEN Y=Y1+(Y2-Y1)*(VX2-X1)/(X2-X1):X=VX2:GOTO *CLIP2
470  IF C% AND 4 THEN X=X1+(X2-X1)*(VY1-Y1)/(Y2-Y1):Y=VY1:GOTO *CLIP2
480  IF C% AND 8 THEN X=X1+(X2-X1)*(VY2-Y1)/(Y2-Y1):Y=VY2
490 *CLIP2
500  IF C%=C1% THEN X1=X:Y1=Y:GOSUB *CHECK:C1%=CVAL% ELSE X2=X:Y2=Y:GO
SUB *CHECK:C2%=CVAL%
510  GOTO *CLIP1
520 '
530 *CHECK
540  CVAL%=0
550  IF X<VX1 THEN CVAL%=1
560  IF VX2<X THEN CVAL%=CVAL%+2
570  IF Y<VY1 THEN CVAL%=CVAL%+4
580  IF VY2<Y THEN CVAL%=CVAL%+8
590  RETURN
```

8.3 3次元変換

さて，お待ちかねの3次元グラフィックスですが，3次元でも2次元でも表示以外の操作は，ほぼ同じ事を行います．違うのは座標軸が増えただけです．2次元変換を理解できれば，3次元でもまごつく事はありません．では順に平行移動から説明しましょう．

8.3.1 平行移動

3次元の平行移動でも2次元と考え方は同じです．ただx軸，y軸の他にz軸が増えただけですので，2次元での平行移動の式にz軸方向への操作を付け加えます．

$$\begin{cases} X = x + T_x \\ Y = y + T_y \\ Z = z + T_z \end{cases}$$

この3つの式により点（x，y，z）はx軸方向にT_x，y軸方向にT_y，z軸方向にT_z移動します．

ここでも2次元変換の時と同様に，ベクトルと行列を使ってこの式を表すことができます．

$$[X \quad Y \quad Z \quad 1] = [x \quad y \quad z \quad 1] \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 0 \\ T_x & T_y & T_z & 1 \end{bmatrix}$$

8.3.2 拡大・縮小

3次元での拡大・縮小は，次の3式で行います．

$$\begin{cases} X = x * S_x \\ Y = y * S_y \\ Z = z * S_z \end{cases}$$

この式によりx，y，zはそれぞれS_x倍，S_y倍，S_z倍されますが，この拡大・縮小は原点(0，0，0)を中心に行います．

$$[X \quad Y \quad Z \quad 1] = [x \quad y \quad z \quad 1] \begin{bmatrix} S_x & 0 & 0 & 0 \\ 0 & S_y & 0 & 0 \\ 0 & 0 & S_z & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix}$$

8.3.3 回転

3次元での回転も基本的には2次元と同じですが，3次元では回転軸が3つあります．

図9　回転とその回転軸

図9の様にx，y，z軸のそれぞれについての回転があり，x軸を中心とした回転をピッチ，y軸を中心とした回転をヘディング，z軸を中心とした回転をバンクとも呼びます．

では，それぞれの回転の式を次に示します．

(i)　x軸中心の回転(ピッチ)　　$[X \quad Y \quad Z \quad 1] = [x \quad y \quad z \quad 1] \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & \cos P & \sin P & 0 \\ 0 & -\sin P & \cos P & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix}$

$$\begin{cases} X = x \\ Y = y\cos P - z\sin P \\ Z = y\sin P + z\cos P \end{cases}$$

（ii） y軸中心の回転(ヘディング) [X Y Z 1]=[x y z 1]$\begin{bmatrix} \mathrm{conH} & 0 & -\sin H & 0 \\ 0 & 1 & 0 & 0 \\ \sin H & 0 & \cos H & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix}$

$$\begin{cases} X = x\cos H + z\sin H \\ Y = y \\ Z = -x\sin H + z\cos H \end{cases}$$

（iii） z軸中心の回転(バンク) [X Y Z 1]=[x y z 1]$\begin{bmatrix} \cos B & \sin B & 0 & 0 \\ -\sin B & \cos B & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix}$

$$\begin{cases} X = x\cos B - y\sin B \\ Y = x\sin B + y\cos B \\ Z = z \end{cases}$$

さて，この3種類の式の組み合わせにより任意の回転ができますが，ここで回転を行う順番が問題になります．例えばx軸中心に90°，y軸中心に90°回転する時，x軸とy軸のどちらの回転を先に行うかで，回転後の位置は別のものになります（図10）.

z軸中心に90°回転

x軸中心に90°回転

x軸中心の回転を先に実行した場合

z軸中心の回転を先にした場合

図10 回転軸を選ぶ順番とその影響

これは行列演算で考えると当然の事で，A，Bを行列とした時，

A・B＝B・A

が成立するのはある特定の条件の時だけで，一般には等しくない事が証明されているのです．

ですから回転に限らず，例えば平行移動と回転を組み合わせた場合にも，この事は問題になる訳です．あなたが3次元グラフィックスで図形の移動や回転，拡大・縮小をする時には，その順序に十分注意してください．

8.3.4 視野とクリッピング

2次元変換でクリッピングについて説明しましたが，ここで説明する3次元でのクリッピング

は，少し異なります．

2次元でのクリッピングは，単に長方形に定めた表示領域内にだけ表示を行う事でしたが，3次元では視野ピラミッドと言う考え方をします．例えば，ある物を写真にとる時，その物全体が写りきらない時には，後にさがりますね．これは同じ物体を見る時，遠くの方が見かけ上の大きさが小さくなる事実を無意識のうちに理解しているからですが，3次元グラフィックでも立体をそれらしく描くために，これと同じ事をしなければなりません．

そこで図11の様な四角錐を考え，この内部だけが見える空間を考えます．この四角錐を視野ピラミッドと呼びますが，3次元でのクリッピングとは，この視野ピラミッド内のものだけを表示する様な操作の事を言います．

では，この視野ピラミッドにおけるクリッピングについて説明しましょう．説明を簡単にするため，視野ピラミッドは，$z=x$，$z=-x$，$z=y$，$z=-y$で表される4つの平面によって囲まれた空間とします（図12）．

図11　視野ピラミッド

図12　視野ピラミッドとクリッピング

さて，2次元でのクリッピングを思い出してください．2次元のクリッピングでは，$x = CX_1$，$x = CX_2$，$y = CX_1$，$y = Cx_2$ の4つの直線に囲まれた長方形を表示領域に設定して，直線の両端の点のそれぞれについて領域内にあるかを調べました．そしてその時の条件判断は，

$$X < CX_1 \quad X > CX_2 \quad Y < CY_1 \quad Y > CY_2$$

この4つでした．

視野ピラミッドにおけるクリッピングでの条件判断もこれに似ており，次に挙げる4つの条件です．ここでも2次元の場合と同様に，どれか1つの条件でも満たしている時には，その点(x，y，z）は視野ピラミッドの外にあります．

$$x < -z \quad x > z \quad y < -z \quad y > z$$

そしてこの条件を使ってクリッピングをしていきますが，実は，視野ピラミッドにおけるクリッピングも，2次元でのクリッピングと，アルゴリズムはほとんど同じなのです．違うのは前に挙げた条件判断と，クリッピングを行って求める点が，直線と直線の交点から，直線と平面の交点に変わっただけなのです．ですから，ここではクリッピングのサブルーチンだけを示すだけにして，アルゴリズムについては2次元でのクリッピングを参照して頂く事にします．

リスト7

```
1000 *CLIP.3D                         直線の両端について,視野ピラミッド内かどうかチェック
1010  X=X1 : Y=Y1 : Z=Z1 : GOSUB *CHECK.3D : C1%=CVAL%}
1020  X=X2 : Y=Y2 : Z=Z2 : GOSUB *CHECK.3D : C2%=CVAL%}
1030 *CLIP1.3D
1040  IF C1%=0 AND C2%=0 THEN *LINE.3D ←直線の両端が共に視野ピラミッド内にある
1050  IF C1% AND C2% THEN RETURN ←両点が共に同じ条件を満たさない
1060  IF C1% THEN C%=C1% ELSE C%=C2%
1070  IF C% AND 1 THEN W=(Z1+X1)/((X1-X2)-(Z2-Z1)) : X=-Z : Y=W*(Y2-Y1
)+Y1 : Z=W*(Z2-Z1)+Z1 : GOTO *CLIP2.3D 平面x =-z との交点を求める
1080  IF C% AND 2 THEN W=(Z1-X1)/((X2-X1)-(Z2-Z1)) : X=Z : Y=W*(Y2-Y1)
+Y1 : Z=W*(Z2-Z1)+Z1 : GOTO *CLIP2.3D  平面x=z との交点を求める
1090  IF C% AND 4 THEN W=(Z1+Y1)/((Y1-Y2)-(Z2-Z1)) : Y=-Z : X=W*(X2-X1
)+X1 : Z=W*(Z2-Z1)+Z1 : GOTO *CLIP2.3D 平面y=-zとの交点を求める
1100  IF C% AND 8 THEN W=(Z1-Y1)/((Y2-Y1)-(Z2-Z1)) : Y=Z : X=W*(X2-X1)
+X1 : Z=W*(Z2-Z1)+Z1                  平面y=  zとの交点を求める
1110 *CLIP2.3D
1120  IF C%=C1% THEN X1=X:Y1=Y:Z1=Z:GOSUB *CHECK.3D:C1%=CVAL% ELSE X2=
X:Y2=Y:Z2=Z:GOSUB *CHECK.3D:C2%=CVAL%  ↖求めた交点について再びチェック
1130  GOTO *CLIP1.3D
1140 '
1150 *CHECK.3D 視野ピラミッド内かどうかのチェック
1160  CVAL%=0
1170  IF X<-Z THEN CVAL%=1
1180  IF Z<X THEN CVAL%=CVAL%+2
1190  IF Y<-Z THEN CVAL%=CVAL%+4
1200  IF Z<Y THEN CVAL%=CVAL%+8
1210  RETURN
1220 '
1230 *LINE.3D
1240  ' this subroutine draw line in 3D
1250  '
1260  ' line (x1,y1,z1)-(x2,y2,z2)
1270 
1280  RETURN
```

8.3.5 透視変換

立体を2次元の平面上に描く方法として透視図があります．そして，この透視法として一般に図13に示す単点透視，2点透視，3点透視の3種類が知られています．

単点透視　2点透視　3点透視

図13　透視図法

3次元グラフィックでも，平面上に3次元での立体を投影するのに，これらの方法を使うのですが，ここでは，最も簡単な単点透視を行いましょう．

3次元物体の見かけの大きさは，遠くはなれた物ほど小さくなりますが，一般にその大きさは視点からの距離に反比例する事が分かっています．そこで視点を原点にあるとすると，点（x，y，z）の見かけ上の位置（X，Y）は，

$$
\begin{cases} X = x / z \\ Y = y / z \end{cases}
$$

で示されます．あまりにも簡単すぎると思われるかもしれませんが，実用上はこれを知っていれば十分なのです．

8.4 3次元パッケージ

これまで2次元変換，3次元変換と3次元グラフィックスへのアプローチとして基礎的な事を説明してきました．そこで今までの説明のまとめとして，ここで3次元パッケージを（3次元物体を視点を変えて，2次元であるCRT上に投影するパッケージ）紹介しましょう．

と言っても汎用性のある本格的なものではありません．本格的な3次元グラフィックスを実現するには，今までの説明だけではまだまだ不十分で，すべて説明すれば，それこそ一冊の本ができ上ってしまいます．ですから，ここで紹介するプログラムは，今まで説明した3次元グラフィックスの基礎を理解するためのサンプル・プログラムと思ってください．

リスト8

```
1000 '
1010 ' 3D graphic
1020 '
1030  N=27 : DIM P(N,3)
1040  RESTORE *DATA.3D
1050  FOR I=0 TO N : FOR J=0 TO 3 : READ P(I,J) : NEXT : NEXT
1060  WIDTH 80,25 : SCREEN 0,3 : CLS 2 : SCREEN 0,0
1070  ON ERROR GOTO *TRAP
1080  ON STOP GOSUB *BREAK : STOP ON
1090  CONSOLE 20,5,0,1 : CLS
1100 '
1110 '  set view port & window
1120  VX1=0 : VX2=639 : VY1=0 : VY2=199 ←ビューポートの設定
1130  WX1=-160 : WY1=-100 : WX2=160 : WY2=100 ←ウィンドウの設定
1140 '                         ←視点距離の設定
1150  TX=0 : TY=0 : TZ=2000 : PTH=1000 ←パースの設定
1160  COL=3 ←表示色の設定
1170 *LOOP    メインルーチン
1180  PRINT
1190  INPUT "Z rotation (Deg.) bank    = ";B : B=3.1416*B/180 ┐
1200  INPUT "Y rotation (Deg.) heading = ";H : H=3.1416*H/180 ├回転角の入力
1210  INPUT "X rotation (Deg.) pitch   = ";P : P=3.1416*P/180 ┘
1220  SCREEN ,1 : CLS 2 : SCREEN ,0
1230  FOR I=0 TO N
1240   GOSUB *MOVE.3D
1250   IF P(I,0) THEN X0=X:Y0=Y:Z0=Z : GOTO *SKIP
1260   X1=X0:Y1=Y0:Z1=Z0 : X2=X:Y2=Y:Z2=Z : X0=X:Y0=Y:Z0=Z
1270   GOSUB *CLIP.3D
1280  *SKIP
1290  NEXT
1300  GOTO *LOOP
1310 '
1320 *TRAP
1330  ON ERROR GOTO 0
1340 '
1350 *BREAK
1360  CONSOLE 0,25,0,1 : STOP OFF
1370  END
1380 '
1390 *MOVE.3D
1400  X=P(I,1) : Y=P(I,2) : Z=P(I,3)
1410 '  rotate bank
1420  W=X*COS(B)-Y*SIN(B) : Y=X*SIN(B)+Y*COS(B) : X=W ┐
1430 '  rotate heading                                  │
1440  W=X*COS(H)+Z*SIN(H) : Z=-X*SIN(H)+Z*COS(H): X=W  ├立体の回転
1450 '  rotate pitch                                    │
1460  W=Y*COS(P)-Z*SIN(P) : Z=Y*SIN(P)+Z*COS(P) : Y=W ┘
1470 '
1480  X=X+TX : Y=Y+TY : Z=Z+TZ ←視野との距離分だけ平行移動
1490  RETURN
1500 '
1510 *DATA.3D 立体のデータ
1520 DATA 1,-100,-100,-100 ──→点を直線で結んで表示する時の
1530 DATA 0, 100,-100,-100     最初の点は1，その他は0
1540 DATA 0, 100,-100, 100
1550 DATA 0,-100,-100, 100
1560 DATA 0,-100,-100,-100
1570 DATA 0,-100, 100,-100
1580 DATA 0, 100, 100,-100
1590 DATA 0, 100,-100,-100
1600 DATA 1, 100, 100,-100
1610 DATA 0, 100, 100, 100
1620 DATA 0, 100,-100, 100
1630 DATA 1, 100, 100, 100
1640 DATA 0,   0, 100, 100
1650 DATA 0,   0, 100,   0
1660 DATA 0,-100, 100,   0
1670 DATA 0,-100, 100,-100
1680 DATA 1,-100, 100,   0
1690 DATA 0,-100,   0,   0
1700 DATA 0,   0,   0,   0
1710 DATA 0,   0,   0, 100
1720 DATA 0,-100,   0, 100
1730 DATA 0,-100,   0,   0
```

```
1740 DATA 1,   0, 100,   0
1750 DATA 0,   0,   0,   0
1760 DATA 1,   0, 100, 100
1770 DATA 0,   0,   0, 100
1780 DATA 1,-100,   0, 100
1790 DATA 0,-100,-100, 100 ←座標 (x, y, z)
1800 '
1810 *CLIP.3D ←――――――――――視野ピラミッドとのクリッピング・サブルーチン
1820  X=X1 : Y=Y1 : Z=Z1 : GOSUB *CHECK.3D : C1%=CVAL%
1830  X=X2 : Y=Y2 : Z=Z2 : GOSUB *CHECK.3D : C2%=CVAL%
1840 *CLIP1.3D
1850  IF C1%=0 AND C2%=0 THEN *LINE.3D
1860  IF C1% AND C2% THEN RETURN
1870  IF C1% THEN C%=C1% ELSE C%=C2%
1880  IF C% AND 1 THEN W=(Z1+X1)/((X1-X2)-(Z2-Z1)) : X=-Z : Y=W*(Y2-Y1
)+Y1 : Z=W*(Z2-Z1)+Z1 : GOTO *CLIP2.3D
1890  IF C% AND 2 THEN W=(Z1-X1)/((X2-X1)-(Z2-Z1)) : X=Z : Y=W*(Y2-Y1)
+Y1 : Z=W*(Z2-Z1)+Z1 : GOTO *CLIP2.3D
1900  IF C% AND 4 THEN W=(Z1+Y1)/((Y1-Y2)-(Z2-Z1)) : Y=-Z : X=W*(X2-X1
)+X1 : Z=W*(Z2-Z1)+Z1 : GOTO *CLIP2.3D
1910  IF C% AND 8 THEN W=(Z1-Y1)/((Y2-Y1)-(Z2-Z1)) : Y=Z : X=W*(X2-X1)
+X1 : Z=W*(Z2-Z1)+Z1
1920 *CLIP2.3D
1930  IF C%=C1% THEN X1=X:Y1=Y:Z1=Z:GOSUB *CHECK.3D:C1%=CVAL% ELSE X2=
X:Y2=Y:Z2=Z:GOSUB *CHECK.3D:C2%=CVAL%
1940  GOTO *CLIP1.3D
1950
1960 *CHECK.3D 視野ピラミッド内かどうかのチェック
1970  CVAL%=0
1980  IF X<-Z THEN CVAL%=1
1990  IF Z<X THEN CVAL%=CVAL%+2
2000  IF Y<-Z THEN CVAL%=CVAL%+4
2010  IF Z<Y THEN CVAL%=CVAL%+8
2020  RETURN
2030 '
2040 *LINE.3D  1点透視の変換
2050  X1=PTH*X1/Z1 : Y1=PTH*Y1/Z1 : X2=PTH*X2/Z2 : Y2=PTH*Y2/Z2
2060
2070 *WINDOW.2D ウィンドウからビューポートへの座標変換
2080  X1=((VX2-VX1)*(X1-WX1)/(WX2-WX1)+VX1)
2090  Y1=(VY2-VY1)*(Y1-WY1)/(WY2-WY1)+VY1
2100  X2=((VX2-VX1)*(X2-WX1)/(WX2-WX1)+VX1)
2110  Y2=(VY2-VY1)*(Y2-WY1)/(WY2-WY1)+VY1
2120 '
2130 *CLIP.2D ビューポートでのクリッピング
2140  X=X1 : Y=Y1 : GOSUB *CHECK.2D : C1%=CVAL%
2150  X=X2 : Y=Y2 : GOSUB *CHECK.2D : C2%=CVAL%
2160 *CLIP1.2D
2170  IF C1%=0 AND C2%=0 THEN *LINE.2D
2180  IF C1% AND C2% THEN RETURN
2190  IF C1% THEN C%=C1% ELSE C%=C2%
2200  IF C% AND 1 THEN Y=Y1+(Y2-Y1)*(VX1-X1)/(X2-X1):X=VX1:GOTO *CLIP2
.2D
2210  IF C% AND 2 THEN Y=Y1+(Y2-Y1)*(VX2-X1)/(X2-X1):X=VX2:GOTO *CLIP2
.2D
2220  IF C% AND 4 THEN X=X1+(X2-X1)*(VY1-Y1)/(Y2-Y1):Y=VY1:GOTO *CLIP2
.2D
2230  IF C% AND 8 THEN X=X1+(X2-X1)*(VY2-Y1)/(Y2-Y1):Y=VY2
2240 *CLIP2.2D
2250  IF C%=C1% THEN X1=X:Y1=Y:GOSUB *CHECK.2D:C1%=CVAL% ELSE X2=X:Y2=
Y:GOSUB *CHECK.2D:C2%=CVAL%
2260  GOTO *CLIP1.2D
2270 '
2280 *CHECK.2D ビューポート内かどうかのチェック
2290  CVAL%=0
2300  IF X<VX1 THEN CVAL%=1
2310  IF VX2<X THEN CVAL%=CVAL%+2
2320  IF Y<VY1 THEN CVAL%=CVAL%+4
2330  IF VY2<Y THEN CVAL%=CVAL%+8
2340  RETURN
2350 '
2360 *LINE.2D
2370  LINE (X1,VY2-Y1)-(X2,VY2-Y2),COL ←y軸を上方向に変えて直線を引く
2380  RETURN
```

このプログラムでは，すでに3次元物体のデータが与えられていて，この物体を3次元変換により，いろいろに変化させて表示します．

入力するパラメータは物体を回転させる角度で，ピッチ（x軸中心)，バンク（y軸中心)，ヘディング（z軸中心）の3つの角度を0から360の間で与えます．回転の順番は，バンク，ヘディング，ピッチの順となっています．但し，ここで与える角度の値は，現在からの相対角度ではなく，常に絶対角度ですから注意してください．

ちなみに，この本の表紙の写真は，フラミンゴの3次元データに基づき，このプログラムと同じ様な3次元プログラムにより作成したものです．

8.5 ワイヤ・フレーム

曲面からなる立体を近似的に処理して立体感を簡単に得る方法として，ワイヤ・フレームがあります．

例えば円柱を表示しようとしても，その輪郭が描けるだけでは，立体感はとうてい得る事はできません．光線が当たった様に，微妙な影を付け加えて始めて立体感が得られますが，パーソナル・コンピュータのグラフィックにはこんな芸当はできませんし，また仮に可能だとしてもその処理のために，膨大な計算時間を必要とします．そこで円柱を多角柱で近似し，その表面をます目の様に区切って表示する方法を使ったりします．この表示方法をワイヤ・フレームと言いますが，この方法を用いても立体を多くの点とそれぞれ結ばれた線によって表すのですから，その点の位置のデータを入力するだけでも大変な手間になります．

そこで，ここではある平面図形を1つの軸を中心に回転してできる立体を，ワイヤ・フレームにより表示するプログラムを紹介しましょう．

表示する立体は回転体ですので，回転する前の平面図形のデータさえ与えれば，他の点は計算から容易に求まりますし，平面図形が簡単でも，回転する角度を細かくすれば，結構それらしい立体図形になるものです．

次のプログラムが回転体をワイヤ・フレームにより表示するプログラムですが，基本的な3次元操作は前に述べた3次元パッケージと同じです．違なるのは立体の点のデータを回転により求めているのと，立体の表示をワイヤ・フレームで行っていることだけです．

リスト9

```
1000 '
1010 '  3D graphic (Wired Frame)
1020 '
1030  MAXN=20 : DIV=10←データの最大個数と図形の回転回数の設定
1040  DIM D(MAXN,1),P(MAXN,DIV,2),VP(MAXN,DIV,2),DP(MAXN,2)
1050  CONSOLE 0,25,0,1:WIDTH 80,25
1060  SCREEN 0,3 : CLS 2 : SCREEN 0,0
1070  ON ERROR GOTO *TRAP
1080  ON STOP GOSUB *BREAK : STOP ON
1090  PAI=3.14159 : ASP=.5
1100 '
1110 '  set view port & window
1120  VX1=0 : VX2=639 : VY1=0 : VY2=199←ビューポートの設定
1130  WX1=-160 : WY1=-100 : WX2=160 : WY2=100←ウィンドウの設定
1140 '
1150  TX=0 : TY=0 : TZ=2000 : PTH=2000 ←視点の設定
1160  COL=4←表示色の設定
1170 *MAKEDATA
1180  CLS
1190  INPUT "display sample data (y/n) ";ANS$
1200  SCREEN ,3:CLS 3:SCREEN ,0
1210  LINE(320,0)-(320,199),1:LINE(0,100)-(639,100),1
1220  LOCATE 40,0:PRINT "y";
1230  LOCATE 42,0:PRINT "100";
1240  LOCATE 0,13:PRINT "-160";
1250  LOCATE 79,12:PRINT "x";
1260  LOCATE 77,13:PRINT "160";
```

```
1270  LOCATE 41,24:PRINT "-100";
1280  LOCATE 40,12:PRINT "o";
1290  CONSOLE 15,9:LOCATE 0,15
1300  IF ANS$<>"y" AND ANS$<>"Y" THEN *MKDATA1
1310  RESTORE *SAMPLE:READ N
1320  FOR I=0 TO N
1330    READ X,Y:LINE(319+X/ASP,99-Y)-STEP(2/ASP,2),4,BF
1340    PRINT USING"x =#### , y =####";X,Y:D(I,0)=X:D(I,1)=Y
1350  NEXT
1360  FOR I=0 TO N-1          サンプル・データの読み込みと表示
1370    LINE(320+D(I,0)/ASP,100-D(I,1))-(320+D(I+1,0)/ASP,100-D(I+1,1)
),4
1380  NEXT
1390  GOTO *MKDATA3
1400 '
1410 *MKDATA1
1420  PRINT "please input data .. "
1430  N=-1
1440 *MKDATA2  データ入力
1450   INPUT " x,y ";X,Y
1460   LINE (319+X/ASP,99-Y)-STEP(2/ASP,2),4,BF
1470   INPUT " y(=ok) , e(=last data) , n(=cancel) ";ANS$
1480   IF ANS$="E" THEN ANS$="e"
1490   IF ANS$="Y" THEN ANS$="y"
1500   IF ANS$<>"e" AND ANS$<>"y" THEN LINE (319+X/ASP,99-Y)-STEP(2/AS
P,2),0,BF : GOTO *MKDATA2
1510    N=N+1 : D(N,0)=X:D(N,1)=Y
1520    IF N=0 THEN *MKDATA2
1530    LINE (320+D(N-1,0)/ASP,100-D(N-1,1))-(320+D(N,0)/ASP,100-D(N,1
)),4
1540    IF ANS$<>"e" AND N<MAXN THEN *MKDATA2
1550 *MKDATA3
1560  CONSOLE 0,25
1570  FOR I=0 TO DIV-1
1580   RAD=PAI*2*I/DIV
1590   FOR J=0 TO N                    x軸を中心とした回転体の
1600    P(J,I,0)= D(J,0)               データを作成
1610    P(J,I,1)= D(J,1)*COS(RAD)
1620    P(J,I,2)=-D(J,1)*SIN(RAD)
1630   NEXT
1640  NEXT
1650 '
1660 *LOOP
1670  CLS
1680  INPUT "Z rotation (Deg.) bank    = ";B : B=PAI*B/180
1690  INPUT "Y rotation (Deg.) heading = ";H : H=PAI*H/180  回転角の入力
1700  INPUT "X rotation (Deg.) pitch   = ";P : P=PAI*P/180
1710 '
1720  FOR I=0 TO N
1730   FOR J=0 TO DIV-1
1740    X=P(I,J,0) : Y=P(I,J,1) : Z=P(I,J,2)
1750    GOSUB *MOVE.3D                                 立体の回転及び視点
1760    VP(I,J,0)=X : VP(I,J,1)=Y : VP(I,J,2)=Z
1770   NEXT
1780  NEXT
1790 '
1800  FOR I=0 TO N
1810   FOR J=0 TO 2
1820    VP(I,DIV,J)=VP(I,0,J)
1830   NEXT
1840  NEXT
1850 '
1860 '  display pattern by wired frame
1870 '
1880  SCREEN ,1 : CLS 2 : SCREEN ,0
1890  FOR J=0 TO DIV-1
1900   FOR I=0 TO N-1
1910    X1=VP(I,J,0):Y1=VP(I,J,1):Z1=VP(I,J,2)      回転角ごとの図形の表示
1920    X2=VP(I+1,J,0):Y2=VP(I+1,J,1):Z2=VP(I+1,J,2)
1930    GOSUB *CLIP.3D
1940   NEXT
1950  NEXT
1960 '
```

```
1970  FOR I=0 TO N
1980   FOR J=0 TO DIV-1
1990    X1=VP(I,J,0):Y1=VP(I,J,1):Z1=VP(I,J,2)          回転後の図形の
2000    X2=VP(I,J+1,0):Y2=VP(I,J+1,1):Z2=VP(I,J+1,2)    同一の点を結んで表示
2010    GOSUB *CLIP.3D
2020   NEXT
2030  NEXT
2040 '
2050  INPUT 'make another angle (y/n) ';ANS$
2060  IF ANS$='y' OR ANS$='Y' THEN *LOOP
2070  INPUT 'make another pattern (y/n) ';ANS$
2080  IF ANS$='y' OR ANS$='Y' THEN *MAKEDATA
2090  INPUT 'display by dot (y/n) ';ANS$
2100  IF ANS$<>'y' AND ANS$<>'Y' THEN *BREAK
2110  GOSUB *DOT.3D
2120  GOTO *LOOP
2130 '
2140 *TRAP
2150  ON ERROR GOTO 0
2160 '
2170 *BREAK
2180  STOP OFF : END
2190 '
2200 *MOVE.3D
2210 ' rotate bank
2220  W=X*COS(B)-Y*SIN(B) : Y=X*SIN(B)+Y*COS(B) : X=W
2230 ' rotate heading
2240  W=X*COS(H)+Z*SIN(H) : Z=-X*SIN(H)+Z*COS(H): X=W   立体の回転
2250 ' rotate pitch
2260  W=Y*COS(P)-Z*SIN(P) : Z=Y*SIN(P)+Z*COS(P) : Y=W
2270 '
2280  X=X+TX : Y=Y+TY : Z=Z+TZ ←視点の操作
2290  RETURN
2300 '
2310 *SAMPLE サンプル・データ
2320 DATA 6  ←データの数 (データの個数)－1
2330 DATA -75, 40
2340 DATA -65, 22
2350 DATA -55, 5     パフェグラスの
2360 DATA -25, 5
2370 DATA  25, 22    2次元データ (x, y)
2380 DATA  75, 50
2390 DATA  75, 46
2400 '
2410 *CLIP.3D 視野ピラミッドでのクリッピング・ルーチン
2420  X=X1 : Y=Y1 : Z=Z1 : GOSUB *CHECK.3D : C1%=CVAL%
2430  X=X2 : Y=Y2 : Z=Z2 : GOSUB *CHECK.3D : C2%=CVAL%
2440 *CLIP1.3D
2450  IF C1%=0 AND C2%=0 THEN *LINE.3D
2460  IF C1% AND C2% THEN RETURN
2470  IF C1% THEN C%=C1% ELSE C%=C2%
2480  IF C% AND 1 THEN W=(Z1+X1)/((X1-X2)-(Z2-Z1)) : X=-Z : Y=W*(Y2-Y1
)+Y1 : Z=W*(Z2-Z1)+Z1 : GOTO *CLIP2.3D
2490  IF C% AND 2 THEN W=(Z1-X1)/((X2-X1)-(Z2-Z1)) : X=Z : Y=W*(Y2-Y1)
+Y1 : Z=W*(Z2-Z1)+Z1 : GOTO *CLIP2.3D
2500  IF C% AND 4 THEN W=(Z1+Y1)/((Y1-Y2)-(Z2-Z1)) : Y=-Z : X=W*(X2-X1
)+X1 : Z=W*(Z2-Z1)+Z1 : GOTO *CLIP2.3D
2510  IF C% AND 8 THEN W=(Z1-Y1)/((Y2-Y1)-(Z2-Z1)) : Y=Z : X=W*(X2-X1)
+X1 : Z=W*(Z2-Z1)+Z1
2520 *CLIP2.3D
2530  IF C%=C1% THEN X1=X:Y1=Y:Z1=Z:GOSUB *CHECK.3D:C1%=CVAL% ELSE X2=
X:Y2=Y:Z2=Z:GOSUB *CHECK.3D:C2%=CVAL%
2540  GOTO *CLIP1.3D
2550 '
2560 *CHECK.3D  視野ピラミッド内かどうかのチェック
2570  CVAL%=0
2580  IF X<-Z THEN CVAL%=1
2590  IF Z<X THEN CVAL%=CVAL%+2
2600  IF Y<-Z THEN CVAL%=CVAL%+4
2610  IF Z<Y THEN CVAL%=CVAL%+8
2620  RETURN
2630 '
2640 *LINE.3D   1点透視の変換
```

リスト9

```
2650 'X1=PTH*X1/Z1:Y1=PTH*Y1/Z1:X2=PTH*X2/Z2:Y2=PTH*Y2/Z2
2660 '
2670 *WINDOW.2D ウィンドウからビューポートへの座標変換
2680  X1=(VX2-VX1)*(X1-WX1)/(WX2-WX1)
2690  Y1=(VY2-VY1)*(Y1-WY1)/(WY2-WY1)
2700  X2=(VX2-VX1)*(X2-WX1)/(WX2-WX1)
2710 'Y2=(VY2-VY1)*(Y2-WY1)/(WY2-WY1)
2720 '
2730 *CLIP.2D  ビューポートのクリッピング
2740  X=X1 : Y=Y1 : GOSUB *CHECK.2D : C1%=CVAL%
2750  X=X2 : Y=Y2 : GOSUB *CHECK.2D : C2%=CVAL%
2760 *CLIP1.2D
2770  IF C1%=0 AND C2%=0 THEN *LINE.2D
2780  IF C1% AND C2% THEN RETURN
2790  IF C1% THEN C%=C1% ELSE C%=C2%
2800  IF C% AND 1 THEN Y=Y1+(Y2-Y1)*(VX1-X1)/(X2-X1):X=VX1:GOTO *CLIP2
.2D
2810  IF C% AND 2 THEN Y=Y1+(Y2-Y1)*(VX2-X1)/(X2-X1):X=VX2:GOTO *CLIP2
.2D
2820  IF C% AND 4 THEN X=X1+(X2-X1)*(VY1-Y1)/(Y2-Y1):Y=VY1:GOTO *CLIP2
.2D
2830  IF C% AND 8 THEN X=X1+(X2-X1)*(VY2-Y1)/(Y2-Y1):Y=VY2
2840 *CLIP2.2D
2850  IF C%=C1% THEN X1=X:Y1=Y:GOSUB *CHECK.2D:C1%=CVAL% ELSE X2=X:Y2=
Y:GOSUB *CHECK.2D:C2%=CVAL%
2860 'GOTO *CLIP1.2D
2870 '
2880 *CHECK.2D ビューポート内かどうかのチェック
2890  CVAL%=0
2900  IF X<VX1 THEN CVAL%=1
2910  IF VX2<X THEN CVAL%=CVAL%+2
2920  IF Y<VY1 THEN CVAL%=CVAL%+4
2930  IF VY2<Y THEN CVAL%=CVAL%+8
2940 'RETURN
2950 '
2960 *LINE.2D
2970  LINE (X1,VY2-Y1)-(X2,VY2-Y2),COL
2980 'RETURN
2990 '
3000 '  display pattern by dot
3010 '
3020 *DOT.3D 回転体を点により表示
3030  CLS:INPUT "how many dots ";DOT:DOT=DOT/(N+1)
3040  SCREEN ,1:CLS 2:SCREEN ,0
3050  FOR I=1 TO DOT
3060    RAD=2*PAI*RND ←回転角を乱数で設定
3070    FOR J=0 TO N
3080      X=D(J,0):Y=D(J,1)*COS(RAD):Z=D(J,1)*SIN(RAD)
3090      GOSUB *MOVE.3D
3100      DP(J,0)=X:DP(J,1)=Y:DP(J,2)=Z
3110    NEXT
3120    FOR J=0 TO N-1
3130      R=RND
3140      X=(DP(J+1,0)-DP(J,0))*R+DP(J,0) ┐
3150      Y=(DP(J+1,1)-DP(J,1))*R+DP(J,1) ├直線上の1点を乱数で設定
3160      Z=(DP(J+1,2)-DP(J,2))*R+DP(J,2) ┘
3170      GOSUB *CHECK.3D ←視野ピラミッド内かどうかのチェック
3180      IF CVAL% THEN *DOT1.3D
3190      X=PTH*X/Z:Y=PTH*Y/Z ←1点透視の変換
3200      X=(VX2-VX1)*(X-WX1)/(WX2-WX1)+VX1 ┐
3210      Y=(VY2-VY1)*(Y-WY1)/(WY2-WY1)+VY1 ┘ウィンドウからビューポートへの座標変換
3220      GOSUB *CHECK.2D ←ビューポート内かどうかのチェック
3230      IF CVAL% THEN *DOT1.3D
3240      PSET(X,VY2-Y),COL
3250     *DOT1.3D
3260    NEXT
3270 'NEXT
3280 '
3290  LOCATE 34,25:PRINT "hit any key ";
3300  DUMY$=INPUT$(1)
3310 RETURN
```

サンプルの2次元データ

サンプル・データのワイヤ・フレーム

回転角を変える

ドットにより表示

ブランデー・グラスのデータを入力

ワイヤ・フレームにより表示

回転角を変える

ドットにより表示

プログラムを実行させたら，まず図形を入力します．そして，この図形をx軸を中心に回転させ，立体のデータを作成します．

このプログラムでは回転させる図形のサンプル・データがすでに用意されていますが，これを表示させる場合は，

display sample data (y/n) ?

と表示した時に〝y〟を入力してください．また，これ以外の図形の場合は〝n〟を入力し，サンプル・データを参考にして，回転させる図形を入力します．

次に，立体を表示する時の回転角を入力しますが，これは3次元パッケージと同様に行ってください．

以上の操作により，ワイヤ・フレームによる立体が表示されますが，このプログラムにはワイヤ・フレームの他に，点によって表示する機能があります．

display by dot (y/n) ?

と表示された時にyと答え，次に点の個数を入力すると，この立体を指定した数の点によって描きます．この時に入力する点の個数が少ないと，立体がうまく表示できませんので，最低でも1000個位にしてください．

APPENDIX

　このプログラムは，CP/M(デジタルリサーチ社のOS)上で走るMACRO-80(マイクロソフト社のマクロアセンブラ)によってアセンブルしたものです.以下にマクロアセンブラ特有の命令について解説しておきます.

. PHASE,　. DEPHASE

　これら2つの擬似命令は，アセンブル後のオブジェクトを他の領域に割り付けて実行する事を許します.すなわち，. PHASEブロック内のすべてのラベルは，その領域の先頭番地からの絶対的な値として定義されますが，生成されるコードは，現在割り付け中の領域にロードされます.

注意：このソース・リストは，Z80(ザイログ) ニーモニックで書かれていますから，PC-8801のモニタにより直接ラインアセンブルすることはできません.

```
                ;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;
                ;                                        ;
                ;         PC-8801 Graphic Library        ;
                ;                                        ;
                ;          Sep. 3 82' by cat             ;
                ;                                        ;
                ;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;
                ;
                        .Z80 ——— Z80ニーモニックであることを宣言する
                ;
8000            IMAGE   EQU     8000H   ; image and execute
                                        ;   address
E000            LOAD    EQU     0E000H  ; absolute load
                                        ;   address
                ;
                ; label definitions
                ;
C000            GRTOP   EQU     0C000H  ; G-ram start address
FE80            GRBOT   EQU     0FE80H  ;  ,,    end+1    ,,
0050            GRWB    EQU     80      ; horizontal length
                ;
                ;;; keyboad scan port ;;;
                ;
0000            KBP0    EQU     0       ; key 7654 3210
0001            KBP1    EQU     1       ; key r.,= +*98
0008            KBP8    EQU     8       ; bit 6 is shift key
                ;
                ;;; CRTC (mPD-3301) control prot ;;;
                ;
0050            CRTCP   EQU     50H     ; CRTC paramater
0051            CRTCC   EQU     51H     ; CRTC command
                ;
                ;;; boarder color control port ;;;
                ;
                ; bit 6 : back ground color green
                ; bit 5 :         ,,        red
                ; bit 4 :         ,,        blue
                ;
                ; bit 2 : boarder color green       ,,
                ; bit 1 :         ,,    red         ,,
                ; bit 0 :         ,,    blue        ,,
0052            BCOLP   EQU     52H
                ;
                ;;; picture mixing port ;;;
                ;
                ; output : active low
                ;
                ; bit 3 : G-VRAM3 display enable
                ; bit 2 : G-VRAM2         ,,
                ; bit 1 : G-VRAM1         ,,
                ; bit 0 : TEXT            ,,
                ;
0053            PMIXP   EQU     53H
                ;
                ;;; color pallet select port ;;;
                ;
                ; pallet 0 -- 7 is PORT 54H -- 5BH
                ;  bit 2 is green
                ;  bit 1 is red
                ;  bit 0 is blue
                ;
0054            PPAL    EQU     54H
```

```
                                  ;
                                  ;;; graphic bank select port ;;;
                                  ;
                                  ; input
                                  ; G-VRAM status
                                  ; bit 2 : G-VRAM2 enable
                                  ; bit 1 : G-VRAM1    ,,
                                  ; bit 0 : G-VRAM0    ,,
                                  ;
005C                              GRST    EQU     5CH
                                  ;
                                  ; output
                                  ;
005C                              GREB1   EQU     5CH     ; select bank#1
005D                              GREB2   EQU     5DH     ;   ,,   bank#2
005E                              GREB3   EQU     5EH     ;   ,,   bank#3
005F                              GRDE    EQU     5FH     ; select main bank
                                  ;
                                  ;;; text window offset register deposit port ;;;
                                  ;
                                  ; read/write
                                  ;
0070                              PTWOFF  EQU     70H     ; read/write
0078                              PTWOFI  EQU     78H     ; increment offset
                                  ;
                                  ;;; kanji ROM port ;;;
                                  ;
                                  ; output
                                  ; KJALP : kanji ROM address low byte
                                  ; KJAHP : kanji ROM address high byte
                                  ; KJEP  : kanji ROM data select port
                                  ; KJDP  : kanji ROM data deselect port
                                  ;
                                  ; !!! caution !!!
                                  ;  kanji ROM address is not equal to JIS code
                                  ;
00E8                              KJALP   EQU     0E8H
00E9                              KJAHP   EQU     0E9H
00EA                              KJEP    EQU     0EAH
00EB                              KJDP    EQU     0EBH
                                  ;
2121                              KSPACE  EQU     2121H
                                  ;
                                  ;
                                          .PHASE  LOAD   ……E000Hからアセンブル
E000    F3                        BOOT:   DI                (ORG E000Hとほぼ同じ事)
E001    21 E018                           LD      HL,BOOT9
E004    11 8000                           LD      DE,IMAGE
E007    01 0390                           LD      BC,IMAGEE-IMAGEB
E00A    DB 70                             IN      A,(PTWOFF)
E00C    F5                                PUSH    PSW
E00D    3E 80                             LD      A,IMAGE/256
E00F    D3 70                             OUT     (PTWOFF),A
E011    ED B0                             LDIR
E013    F1                                POP     PSW
E014    D3 70                             OUT     (PTWOFF),A
E016    FB                                EI
E017    C9                                RET
                                  ;
E018                              BOOT9   EQU     $
                                          .DEPHASE   ……前の.PHASE LOADに対応して.PHASEの終了を指示
                                  ;
                                  ;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;
                                  ;
                                          .PHASE  IMAGE  ……8000Hからのメモリに配置されるとしてアセンブル
8000                              IMAGEB  EQU     $        するが実際のオブジェクトはE018Hから生成される
                                  ;
                                  ;;; put kanji pattern to graphic area ;;;
                                  ;
                                  ; *** entry ***
                                  ; KPUT  : for 640*200 dot B/W graphic
                                  ; KPUTH : for 640*400 dot B/W    ,,
                                  ;
                                  ; *** paramaters ***
                                  ; KCODE : in DE ;  the value must be JIS kanji
                                  ;                  code except for half and
                                  ;                  quarter character
                                  ;                   (16 * 16 dot only)
                                  ; KBANK : in ACC;  this paramater select active
                                  ;                  G-bank (only for 640*200 dot
                                  ;                  B/W mode) and value must be
                                  ;                  within 0 -- 2
                                  ; KX    : in B  ;  horizontal location
                                  ; KY    : in C  ;  vertical location
                                  ;
8000    26 00                     KPUTH:  LD      H,0
8002    79                                LD      A,C
8003    D6 0A                             SUB     200/20
8005    38 02                             JR      C,KPUTH1
8007    4F                                LD      C,A
8008    24                                INC     H
```

```
8009    7C              KPUTH1: LD      A,H
                        ;
800A    C6 5C           KPUT:   ADD     A,GREB1
800C    C5                      PUSH    BC
800D    4F                      LD      C,A
800E    EB                      EX      DE,HL
800F    7C                      LD      A,H
8010    FE 30                   CP      30H
8012    7D                      LD      A,L
8013    30 05                   JR      NC,KPUT1
8015    1F                      RRA
8016    E6 30                   AND     30H
8018    18 03                   JR      KPUT2
801A    17              KPUT1:  RLA
801B    E6 C0                   AND     0C0H
801D    47              KPUT2:  LD      B,A
801E    7D                      LD      A,L
801F    17                      RLA
8020    17                      RLA
8021    17                      RLA
8022    E6 F8                   AND     0F8H
8024    17                      RLA
8025    6F                      LD      L,A
8026    7C                      LD      A,H
8027    17                      RLA
8028    E6 3F                   AND     3FH
802A    B0                      OR      B
802B    67                      LD      H,A
802C    22 8377                 LD      (KADR),HL

802F    E1                      POP     HL
8030    CB 24                   SLA     H
8032    5C                      LD      E,H
8033    7D                      LD      A,L
8034    17                      RLA
8035    85                      ADD     A,L
8036    17                      RLA
8037    C6 C0                   ADD     A,GRTOP/256
8039    57                      LD      D,A
803A    AF                      XOR     A
803B    CB 3D                   SRL     L
803D    1F                      RRA
803E    CB 3D                   SRL     L
8040    1F                      RRA
8041    65                      LD      H,L
8042    6F                      LD      L,A
8043    19                      ADD     HL,DE
8044    06 10                   LD      B,16
8046    ED 79                   OUT     (C),A
                        ;
8048    EB              KPUT3:  EX      DE,HL
8049    2A 8377                 LD      HL,(KADR)
804C    7C                      LD      A,H
804D    D3 E9                   OUT     (KJAHP),A
804F    7D                      LD      A,L
8050    D3 E8                   OUT     (KJALP),A
8052    23                      INC     HL
8053    22 8377                 LD      (KADR),HL
8056    AF                      XOR     A
8057    D3 EA                   OUT     (KJEP),A
8059    DB E9                   IN      A,(KJAHP)
805B    67                      LD      H,A
805C    DB E8                   IN      A,(KJALP)
805E    6F                      LD      L,A
805F    AF                      XOR     A
8060    D3 EB                   OUT     (KJDP),A
8062    EB                      EX      DE,HL
8063    72                      LD      (HL),D
8064    23                      INC     HL
8065    73                      LD      (HL),E
8066    11 004F                 LD      DE,GRWB-1
8069    19                      ADD     HL,DE
806A    10 DC                   DJNZ    KPUT3
806C    D3 5F                   OUT     (GRDE),A
806E    C9                      RET
                        ;
                        ;;; clear screen all ;;;
                        ;
                        ; *** entry ***
                        ; CLS   : for 640--200 dot color mode
                        ; CLSS  : for 640--200 dot B/W mode
                        ; CLSH  : for 640--400 dot B/W mode
                        ;
806F    3E 02           CLS:    LD      A,2
8071    CD 807A                 CALL    CLSS
8074    3E 01           CLSH:   LD      A,1
8076    CD 807A                 CALL    CLSS
8079    AF                      XOR     A
807A    C6 5C           CLSS:   ADD     A,GREB1
807C    4F                      LD      C,A
807D    ED 79                   OUT     (C),A
```

```
807F    01 3E7F                    LD      BC,GRBOT-GRTOP-1
8082    11 C001                    LD      DE,GRTOP+1
8085    21 C000                    LD      HL,GRTOP
8088    36 00                      LD      (HL),0
808A    ED B0                      LDIR
808C    D3 5F                      OUT     (GRDE),A
808E    C9                         RET
                          ;
                          ;;; roll up & down screen ;;;
                          ;
                          ; *** entry ***
                          ; ROLUPH,ROLDWH : for 640--400 dot B/W mode
                          ; ROLUP,ROLDW   : for 640--200 dot color mode
                          ;
                          ; *** paramaters ***
                          ; ROLCNT : in ACC ; this paramater is roll up
                          ;                   and down count and value
                          ;                   must be within 1 -- 199
                          ;
808F    CD 815B           ROLUP:   CALL    GVRN
8092    21 C000                    LD      HL,GRTOP
8095    19                         ADD     HL,DE
8096    22 8379                    LD      (ROLP),HL
8099    3E 5C                      LD      A,GREB1
809B    CD 80CD           ROLUP1:  CALL    ROLUP2
809E    3C                         INC     A
809F    FE 5F                      CP      GREB3+1
80A1    20 F8                      JR      NZ,ROLUP1
80A3    C9                         RET
                          ;
80A4    CD 815B           ROLUPH:  CALL    GVRN
80A7    21 C000                    LD      HL,GRTOP
80AA    19                         ADD     HL,DE
80AB    22 8379                    LD      (ROLP),HL
80AE    3E 5C                      LD      A,GREB1
80B0    CD 80E2                    CALL    ROLUP3
80B3    21 C000                    LD      HL,GRTOP
80B6    ED 4B 8381                 LD      BC,(ROLN)
80BA    D3 5D             RLUPH1:  OUT     (GREB2),A
80BC    7E                         LD      A,(HL)
80BD    EB                         EX      DE,HL
80BE    D3 5C                      OUT     (GREB1),A
80C0    77                         LD      (HL),A
80C1    EB                         EX      DE,HL
80C2    23                         INC     HL
80C3    13                         INC     DE
80C4    0B                         DEC     BC
80C5    78                         LD      A,B
80C6    B1                         OR      C
80C7    20 F1                      JR      NZ,RLUPH1
80C9    D3 5F                      OUT     (GRDE),A
80CB    3E 5D                      LD      A,GREB2
                          ;
80CD    CD 80E2           ROLUP2:  CALL    ROLUP3
80D0    62                         LD      H,D
80D1    6B                         LD      L,E
80D2    13                         INC     DE
80D3    4F                         LD      C,A
80D4    ED 79                      OUT     (C),A
80D6    36 00                      LD      (HL),0
80D8    ED 4B 8381                 LD      BC,(ROLN)
80DC    0B                         DEC     BC
80DD    ED B0                      LDIR
80DF    D3 5F                      OUT     (GRDE),A
80E1    C9                         RET
                          ;
80E2    2A 8379           ROLUP3:  LD      HL,(ROLP)
80E5    11 C000                    LD      DE,GRTOP
80E8    4F                         LD      C,A
80E9    ED 79                      OUT     (C),A
80EB    ED 4B 837D                 LD      BC,(ROLC)
80EF    ED B0                      LDIR
80F1    D3 5F                      OUT     (GRDE),A
80F3    C9                         RET
                          ;
                          ;
80F4    CD 815B           ROLDW:   CALL    GVRN
80F7    11 C000                    LD      DE,GRTOP
80FA    19                         ADD     HL,DE
80FB    2B                         DEC     HL
80FC    22 8379                    LD      (ROLP),HL
80FF    3E 5E                      LD      A,GREB3
8101    CD 8134           ROLDW1:  CALL    ROLDW2
8104    3D                         DEC     A
8105    FE 5B                      CP      GREB1-1
8107    20 F8                      JR      NZ,ROLDW1
8109    C9                         RET
                          ;
                          ;
810A    CD 815B           ROLDWH:  CALL    GVRN
810D    11 C000                    LD      DE,GRTOP
```

```
8110    19              ADD     HL,DE
8111    2B              DEC     HL
8112    22 8379         LD      (ROLP),HL
8115    3E 5D           LD      A,GREB2
8117    CD 8149         CALL    ROLDW3
811A    21 FE7F         LD      HL,GRBOT-1
811D    ED 4B 8381      LD      BC,(ROLN)
8121    D3 5C   RLDWH1: OUT     (GREB1),A
8123    7E              LD      A,(HL)
8124    EB              EX      DE,HL
8125    D3 5D           OUT     (GREB2),A
8127    77              LD      (HL),A
8128    EB              EX      DE,HL
8129    2B              DEC     HL
812A    1B              DEC     DE
812B    0B              DEC     BC
812C    78              LD      A,B
812D    B1              OR      C
812E    20 F1           JR      NZ,RLDWH1
8130    D3 5F           OUT     (GRDE),A
8132    3E 5C           LD      A,GREB1
                ;
8134    CD 8149 ROLDW2: CALL    ROLDW3
8137    62              LD      H,D
8138    6B              LD      L,E
8139    1B              DEC     DE
813A    4F              LD      C,A
813B    ED 79           OUT     (C),A
813D    36 00           LD      (HL),0
813F    ED 4B 8381      LD      BC,(ROLN)
8143    0B              DEC     BC
8144    ED B8           LDDR
8146    D3 5F           OUT     (GRDE),A
8148    C9              RET
                ;
8149    2A 8379 ROLDW3: LD      HL,(ROLP)
814C    11 FE7F         LD      DE,GRBOT-1
814F    4F              LD      C,A
8150    ED 79           OUT     (C),A
8152    ED 4B 837D      LD      BC,(ROLC)
8156    ED B8           LDDR
8158    D3 5F           OUT     (GRDE),A
815A    C9              RET
                ;
                ;
815B    A7      GVRN:   AND     A
815C    28 1D           JR      Z,GVRN1
815E    FE C8           CP      200
8160    30 19           JR      NC,GVRN1
8162    26 00           LD      H,0
8164    6F              LD      L,A
8165    54              LD      D,H
8166    5F              LD      E,A
8167    29              ADD     HL,HL
8168    29              ADD     HL,HL
8169    19              ADD     HL,DE
816A    29              ADD     HL,HL
816B    29              ADD     HL,HL
816C    29              ADD     HL,HL
816D    29              ADD     HL,HL
816E    22 8381         LD      (ROLN),HL
8171    EB              EX      DE,HL
8172    21 3E80         LD      HL,GRBOT-GRTOP
8175    ED 52           SBC     HL,DE
8177    22 837D         LD      (ROLC),HL
817A    C9              RET
817B    33      GVRN1:  INC     SP
817C    33              INC     SP
817D    C9              RET
                ;
                ;;; roll left & right screen ;;;
                ;
                ; *** entry ***
                ; ROLLEH,ROLRIH : for 640--400 dot B/W mode
                ; ROLLE,ROLRI   : for 640--200 dot color mode
                ;
                ; *** paramaters ***
                ; ROLCNT : in DE ; this paramater is roll left
                ;                  and right count and value
                ;                  must be within 8 -- 632 and
                ;                  modulo 8 must be zero.
                ;                   if set this paramater 12,
                ;                  exact value become 16.
                ;
817E    CD 8213 ROLLI:  CALL    GHRN
8181    3E 5E           LD      A,GREB3
8183    18 05           JR      ROLLI1
                ;
8185    CD 8213 ROLLIH: CALL    GHRN
8188    3E 5D           LD      A,GREB2
818A    21 C000 ROLLI1: LD      HL,GRTOP
```

```
818D    19                          ADD   HL,DE
818E    22 8379                     LD    (ROLP),HL
8191    21 C050                     LD    HL,GRTOP+GRWB
8194    ED 52                       SBC   HL,DE
8196    22 837B                     LD    (ROLQ),HL
8199    2A 8379          ROLLI2:    LD    HL,(ROLP)
819C    11 C000                     LD    DE,GRTOP
819F    4F                          LD    C,A
81A0    ED 79                       OUT   (C),A
81A2    ED 4B 837D                  LD    BC,(ROLC)
81A6    ED B0                       LDIR
81A8    32 8383                     LD    (ROLWK),A
81AB    2A 837F                     LD    HL,(ROLM)
81AE    EB                          EX    DE,HL
81AF    2A 837B                     LD    HL,(ROLQ)
81B2    0E C8                       LD    C,200
81B4    3A 8381                     LD    A,(ROLN)
81B7    47               ROLLI3:    LD    B,A
81B8    36 00            ROLLI4:    LD    (HL),0
81BA    23                          INC   HL
81BB    10 FB                       DJNZ  ROLLI4
81BD    19                          ADD   HL,DE
81BE    0D                          DEC   C
81BF    20 F6                       JR    NZ,ROLLI3
81C1    3A 8383                     LD    A,(ROLWK)
81C4    3D                          DEC   A
81C5    FE 5B                       CP    GREB1-1
81C7    20 D0                       JR    NZ,ROLLI2
81C9    D3 5F                       OUT   (GRDE),A
81CB    C9                          RET
                         ;
81CC    CD 8213          ROLRI:     CALL  GHRN
81CF    3E 5E                       LD    A,GREB3
81D1    18 05                       JR    ROLRI1
                         ;
81D3    CD 8213          ROLRIH:    CALL  GHRN
81D6    3E 5D                       LD    A,GREB2
81D8    21 FE7F          ROLRI1:    LD    HL,GRBOT-1
81DB    ED 52                       SBC   HL,DE
81DD    22 8379                     LD    (ROLP),HL
81E0    2A 8379          ROLRI2:    LD    HL,(ROLP)
81E3    11 FE7F                     LD    DE,GRBOT-1
81E6    4F                          LD    C,A
81E7    ED 79                       OUT   (C),A
81E9    ED 4B 837D                  LD    BC,(ROLC)
81ED    ED B8                       LDDR
81EF    32 8383                     LD    (ROLWK),A
81F2    2A 837F                     LD    HL,(ROLM)
81F5    EB                          EX    DE,HL
81F6    21 C000                     LD    HL,GRTOP
81F9    0E C8                       LD    C,200
81FB    3A 8381                     LD    A,(ROLN)
81FE    47               ROLRI3:    LD    B,A
81FF    36 00            ROLRI4:    LD    (HL),0
8201    23                          INC   HL
8202    10 FB                       DJNZ  ROLRI4
8204    19                          ADD   HL,DE
8205    0D                          DEC   C
8206    20 F6                       JR    NZ,ROLRI3
8208    3A 8383                     LD    A,(ROLWK)
820B    3D                          DEC   A
820C    FE 5B                       CP    GREB1-1
820E    20 D0                       JR    NZ,ROLRI2
8210    D3 5F                       OUT   (GRDE),A
8212    C9                          RET
                         ;
8213    21 0007          GHRN:      LD    HL,7
8216    19                          ADD   HL,DE
8217    7D                          LD    A,L
8218    CB 3C                       SRL   H
821A    1F                          RRA
821B    CB 3C                       SRL   H
821D    1F                          RRA
821E    CB 3C                       SRL   H
8220    1F                          RRA
8221    26 00                       LD    H,0
8223    6F                          LD    L,A
8224    22 8381                     LD    (ROLN),HL
8227    EB                          EX    DE,HL
8228    21 0050                     LD    HL,GRWB
822B    A7                          AND   A
822C    ED 52                       SBC   HL,DE
822E    22 837F                     LD    (ROLM),HL
8231    21 3E80                     LD    HL,GRBOT-GRTOP
8234    ED 52                       SBC   HL,DE
8236    22 837D                     LD    (ROLC),HL
8239    C9                          RET
```

```
                                ;
                                ;;; move and display graphic cursor ;;;
                                ;
                                ; *** entrys ***
                                ; GCREV  : set paramater and reverse graphic
                                ;           cursor.
                                ;           (if display then erase else display)
                                ; GCMOVE : scan ten-key and if press key then
                                ;          move graphic cursor to the direction
                                ;          and return 255 else return 0.
                                ;
                                ; *** paramater for GCREV ***
                                ; GCX    : in DE ; graphic cursor horizontal
                                ;                  position
                                ; GCY    : in BC ; graphic cursor vertical
                                ;                  position
                                ; GCXM   : in IX ; GCX max value
                                ; GCYM   : in IY ; GCY max value
                                ;
                                ; *** return value of GCMOVE ***
                                ; GCX    : in DE ; graphic cursor horizontal
                                ;                  position
                                ; GCY    : in BC ; graphic cursor vertical
                                ;                  position
                                ; GCVAL  : in ACC; if moved then return 255 else  0

                                ;
                                ; local constance : graphic cursor move step of
                                ;                   unshifted and shifted case
0001                            STEP1   EQU     1 ; unshifted step
0008                            STEP2   EQU     8 ; shifted step
                                ;
823A    ED 53 8385              GCREV:  LD      (GX),DE
823E    ED 43 8387                      LD      (GY),BC
8242    DD 22 8389                      LD      (GXM),IX
8246    FD 22 838B                      LD      (GYM),IY
824A    CD 8334                         CALL    VLINE
824D    C3 82F8                         JP      HLINE
                                ;
8250    AF                      GCMOVE: XOR     A
8251    32 838F                         LD      (GCVAL),A
8254    21 838D                         LD      HL,GS
8257    36 08                           LD      (HL),STEP2
8259    DB 08                           IN      A,(KBP8)
825B    E6 40                           AND     40H             ; scan shift key
825D    28 02                           JR      Z,KBSCAN
825F    36 01                           LD      (HL),STEP1
8261    DB 00                   KBSCAN: IN      A,(KBP0)
8263    2F                              CPL
8264    57                              LD      D,A
8265    DB 01                           IN      A,(KBP1)
8267    2F                              CPL
8268    5F                              LD      E,A
                                ;
8269    01 0000                         LD      BC,0            ; dx=dy=0;
826C    E6 02                           AND     02H             ; scan 9
826E    20 05                           JR      NZ,KBSC1
8270    7A                              LD      A,D
8271    E6 48                           AND     48H             ; scan 36
8273    28 01                           JR      Z,KBSC2
8275    04                      KBSC1:  INC     B
8276    7A                      KBSC2:  LD      A,D
8277    E6 92                           AND     92H             ; scan 147
8279    28 01                           JR      Z,KBSC3
827B    05                              DEC     B
827C    7A                      KBSC3:  LD      A,D
827D    E6 0E                           AND     0EH             ; scan 123
827F    28 01                           JR      Z,KBSC4
8281    0C                              INC     C
8282    7A                      KBSC4:  LD      A,D
8283    E6 80                           AND     80H             ; scan 7
8285    20 05                           JR      NZ,KBSC5
8287    7B                              LD      A,E
8288    E6 03                           AND     03H             ; scan 89
828A    28 01                           JR      Z,KBSC6
828C    0D                      KBSC5:  DEC     C
828D    78                      KBSC6:  LD      A,B
828E    B1                              OR      C
828F    C4 829E                         CALL    NZ,GCMOV2
8292    ED 5B 8385                      LD      DE,(GX)
8296    ED 4B 8387                      LD      BC,(GY)
829A    3A 838F                         LD      A,(GCVAL)
829D    C9                              RET
                                ;
                                ;
829E    3A 838F                 GCMOV2: LD      A,(GCVAL)
82A1    F6 FF                           OR      0FFH
82A3    32 838F                         LD      (GCVAL),A
82A6    78                              LD      A,B
82A7    2A 838D                         LD      HL,(GS)
82AA    EB                              EX      DE,HL
```

```
82AB    21 0000                 LD      HL,0
82AE    A7                      AND     A
82AF    28 15                   JR      Z,GCMOV4
82B1    FA 82B5                 JP      M,GCMOV3
82B4    EB                      EX      DE,HL
82B5    ED 52          GCMOV3:  SBC     HL,DE
82B7    EB                      EX      DE,HL
82B8    2A 8385                 LD      HL,(GX)
82BB    19                      ADD     HL,DE
82BC    EB                      EX      DE,HL
82BD    2A 8389                 LD      HL,(GXM)
82C0    A7                      AND     A
82C1    ED 52                   SBC     HL,DE
82C3    EB                      EX      DE,HL
82C4    30 03                   JR      NC,GCMOV5
82C6    2A 8385        GCMOV4:  LD      HL,(GX)
82C9    C5             GCMOV5:  PUSH    BC
82CA    E5                      PUSH    HL
82CB    CD 8334                 CALL    VLINE
82CE    E1                      POP     HL
82CF    22 8385                 LD      (GX),HL
82D2    CD 8334                 CALL    VLINE
82D5    C1                      POP     BC

82D6    3A 838B                 LD      A,(GYM)
82D9    3C                      INC     A
82DA    47                      LD      B,A
82DB    79                      LD      A,C
82DC    A7                      AND     A
82DD    3A 838D                 LD      A,(GS)
82E0    4F                      LD      C,A
82E1    3A 8387                 LD      A,(GY)
82E4    28 0A                   JR      Z,GCMOV8
82E6    FA 82ED                 JP      M,GCMOV7
82E9    81             GCMOV6:  ADD     A,C
82EA    B8                      CP      B
82EB    38 03                   JR      C,GCMOV8
                       ;
82ED    91             GCMOV7:  SUB     C
82EE    38 F9                   JR      C,GCMOV6
82F0    F5             GCMOV8:  PUSH    AF
82F1    CD 82F8                 CALL    HLINE
82F4    F1                      POP     AF
82F5    32 8387                 LD      (GY),A
                       ;
82F8    2A 8387        HLINE:   LD      HL,(GY)
                                REPT    4
                                ADD     HL,HL
                                ENDM
82FB    29          +           ADD     HL,HL
82FC    29          +           ADD     HL,HL
82FD    29          +           ADD     HL,HL
82FE    29          +           ADD     HL,HL
82FF    54                      LD      D,H
8300    5D                      LD      E,L
8301    29                      ADD     HL,HL
8302    29                      ADD     HL,HL
8303    19                      ADD     HL,DE
8304    11 C000                 LD      DE,GRTOP
8307    19                      ADD     HL,DE
8308    E5                      PUSH    HL
8309    ED 4B 8389              LD      BC,(GXM)
830D    79                      LD      A,C
                                REPT    3
                                SRL     B
                                RR      C
                                ENDM
830E    CB 38       +           SRL     B
8310    CB 19       +           RR      C
8312    CB 38       +           SRL     B
8314    CB 19       +           RR      C
8316    CB 38       +           SRL     B
8318    CB 19       +           RR      C
831A    E6 07                   AND     7
831C    16 00                   LD      D,0
831E    5F                      LD      E,A
831F    21 8367                 LD      HL,HBITT
8322    19                      ADD     HL,DE
8323    41                      LD      B,C
8324    4E                      LD      C,(HL)
8325    E1                      POP     HL
8326    D3 5C                   OUT     (GREB1),A
8328    7E             HLINE1:  LD      A,(HL)
8329    2F                      CPL
832A    77                      LD      (HL),A
832B    23                      INC     HL
832C    10 FA                   DJNZ    HLINE1
832E    79                      LD      A,C
832F    AE                      XOR     (HL)
8330    77                      LD      (HL),A
8331    D3 5F                   OUT     (GRDE),A
```

RERTとENDMの間のADD HL, HLを4回繰り返してオブジェクトを生成する

REPTとENDMの間のSRL BとRR Cを3回繰り返してオブジェクト生成する

```
8333    C9                        RET
                          ;
8334    2A 8385           VLINE:  LD      HL,(GX)
8337    7D                        LD      A,L
                                  REPT    3
                                  SRL     H          REPTとENDMの間のSRL HとRR Lを
                                  RR      L          3回繰り返してオブジェクトを生成する
                                  ENDM
8338    CB 3C         +           SRL     H
833A    CB 1D         +           RR      L
833C    CB 3C         +           SRL     H
833E    CB 1D         +           RR      L
8340    CB 3C         +           SRL     H
8342    CB 1D         +           RR      L
8344    11 C000                   LD      DE,GRTOP
8347    19                        ADD     HL,DE
8348    EB                        EX      DE,HL
8349    21 836F                   LD      HL,VBITT
834C    E6 07                     AND     7
834E    06 00                     LD      B,0
8350    4F                        LD      C,A
8351    09                        ADD     HL,BC
8352    4E                        LD      C,(HL)
8353    EB                        EX      DE,HL
8354    11 0050                   LD      DE,GRWB
8357    3A 838B                   LD      A,(GYM)
835A    3C                        INC     A
835B    47                        LD      B,A
835C    D3 5C                     OUT     (GREB1),A
835E    79                VLINE1: LD      A,C
835F    AE                        XOR     (HL)
8360    77                        LD      (HL),A
8361    19                        ADD     HL,DE
8362    10 FA                     DJNZ    VLINE1
8364    D3 5F                     OUT     (GRDE),A
8366    C9                        RET
                          ;
                          ;
8367    80 C0 E0 F0       HBITT:  DEFB    80H,0C0H,0E0H,0F0H
836B    F8 FC FE FF               DEFB    0F8H,0FCH,0FEH,0FFH
836F    80 40 20 10       VBITT:  DEFB    80H,40H,20H,10H,8,4,2,1
8373    08 04 02 01
                          ;
8377    0000              KADR:   DEFW    0
                          ;
8379    0000              ROLP:   DEFW    0
837B    0000              ROLQ:   DEFW    0
837D    0000              ROLC:   DEFW    0
837F    0000              ROLM:   DEFW    0
8381    0000              ROLN:   DEFW    0
8383    0000              ROLWK:  DEFW    0
                          ;
8385    0140              GX:     DEFW    320
8387    0063              GY:     DEFW    99
8389    027F              GXM:    DEFW    639
838B    00C7              GYM:    DEFW    199
838D    0001              GS:     DEFW    STEP1
838F    00                GCVAL:  DEFB    0

8390                      IMAGEE  EQU     $ ……. PHASE IMAGEに対応して. PHASEの終了を指示
                                  .DEPHASE
                          ;
                          ;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;
                          ;
                                  .PHASE  IMAGEE-IMAGEB+BOOT9 ……E3A8Hからアセンブルする
E3A8                              DEFS    $/256*256-$
                          ;
                          ;
E400    F3                ENTRY:  DI
E401    FE 02                     CP      2
E403    20 2F                     JR      NZ,EXIT
E405    22 E5E1                   LD      (VARPTR),HL
E408    AF                        XOR     A
E409    32 E5E3                   LD      (USRVAL),A
E40C    23                        INC     HL
E40D    7E                        LD      A,(HL)
E40E    A7                        AND     A
E40F    20 23                     JR      NZ,EXIT
E411    57                        LD      D,A
E412    2B                        DEC     HL
E413    7E                        LD      A,(HL)
E414    FE 13                     CP      LIBMAX+1
E416    30 1C                     JR      NC,EXIT
E418    87                        ADD     A,A
E419    87                        ADD     A,A
E41A    5F                        LD      E,A
E41B    21 E452                   LD      HL,JMPTBL
E41E    19                        ADD     HL,DE
E41F    5E                        LD      E,(HL)
E420    23                        INC     HL
E421    56                        LD      D,(HL)
```

```
E422    23                      INC     HL
E423    4E                      LD      C,(HL)
E424    23                      INC     HL
E425    46                      LD      B,(HL)
E426    C5                      PUSH    BC
E427    CD E446                 CALL    SELBNK
E42A    CD E450                 CALL    JUMP
E42D    E1                      POP     HL
E42E    CD E451                 CALL    JUMP1
E431    CD E440                 CALL    DSLBNK
E434    2A E5E1         EXIT:   LD      HL,(VARPTR)
E437    3A E5E3                 LD      A,(USRVAL)
E43A    77                      LD      (HL),A
E43B    23                      INC     HL
E43C    36 00                   LD      (HL),0
E43E    FB                      EI
E43F    C9                      RET
                        ;
E440    3A E53D         DSLBNK: LD      A,(SVOFF)
E443    D3 70                   OUT     (PTWOFF),A
E445    C9              PARM0:  RET
                        ;
E446    DB 70           SELBNK: IN      A,(PTWOFF)
E448    32 E53D                 LD      (SVOFF),A
E44B    3E 80                   LD      A,IMAGE/256
E44D    D3 70                   OUT     (PTWOFF),A
E44F    C9                      RET
                        ;
E450    EB              JUMP:   EX      DE,HL
E451    E9              JUMP1:  JP      (HL)
                        ;
                        ; graphic library jump table
                        ;
E452    E445 806F       JMPTBL: DEFW    PARM0,CLS       ;  0
E456    E49A 807A               DEFW    PARM1,CLSS      ;  1
E45A    E445 8074               DEFW    PARM0,CLSH      ;  2
E45E    E49A 808F               DEFW    PARM1,ROLUP     ;  3
E462    E49A 80A4               DEFW    PARM1,ROLUPH    ;  4
E466    E49A 80F4               DEFW    PARM1,ROLDW     ;  5
E46A    E49A 810A               DEFW    PARM1,ROLDWH    ;  6
E46E    E4BD 817E               DEFW    PARM2,ROLLI     ;  7
E472    E4BD 8185               DEFW    PARM2,ROLLIH    ;  8
E476    E4BD 81CC               DEFW    PARM2,ROLRI     ;  9
E47A    E4BD 81D3               DEFW    PARM2,ROLRIH    ; 10
E47E    E4B7 E53E               DEFW    PARM3,PROT      ; 11
E482    E4B7 E57C               DEFW    PARM3,PROTX     ; 12
E486    E4B7 E59B               DEFW    PARM3,PROTY     ; 13
E48A    E49F 823A               DEFW    PARM4,GCREV     ; 14
E48E    8250 E52D               DEFW    GCMOVE,PARM5    ; 15
E492    E445 E4C4               DEFW    PARM0,KPUTL     ; 16
E496    E445 E4CD               DEFW    PARM0,KPUTLH    ; 17
                        ;
0012                    LIBMAX  EQU     ($-JMPTBL)/4
                        ;
                        ;
E49A                    PARM1:  ; for ROLUP,ROLUPH,ROLDW,ROLDWH
                                ;     CLSS
E49A    2A E5F8                 LD      HL,(ARG0)
E49D    7E                      LD      A,(HL)
E49E    C9                      RET
                        ;
E49F                    PARM4:  ; for GCREV
E49F    2A E5FC                 LD      HL,(ARG2)
E4A2    CD E4C0                 CALL    PARMG
E4A5    DD 21 0000              LD      IX,0
E4A9    DD 19                   ADD     IX,DE
                        ;
E4AB    2A E5FE                 LD      HL,(ARG3)
E4AE    CD E4C0                 CALL    PARMG
E4B1    FD 21 0000              LD      IY,0
E4B5    FD 19                   ADD     IY,DE
                        ;
E4B7                    PARM3:  ; for PROT,PROTX,PROTY
E4B7    2A E5FA                 LD      HL,(ARG1)
E4BA    4E                      LD      C,(HL)
E4BB    23                      INC     HL
E4BC    46                      LD      B,(HL)
                        ;
E4BD                    PARM2:  ; for   ROLLI,ROLLIH,ROLRI,ROLRIH
E4BD    2A E5F8                 LD      HL,(ARG0)
E4C0    5E              PARMG:  LD      E,(HL)
E4C1    23                      INC     HL
E4C2    56                      LD      D,(HL)
E4C3    C9                      RET
                        ;
E4C4    2A E5FC         KPUTL:  LD      HL,(ARG2)
E4C7    7E                      LD      A,(HL)
E4C8    21 800A                 LD      HL,KPUT
E4CB    18 03                   JR      KPUTL1
E4CD    21 8000         KPUTLH: LD      HL,KPUTH
E4D0    32 E5DA         KPUTL1: LD      (KBANK),A
```

```
E4D3     22 E5DD                 LD       (KVECT),HL
E4D6     2A E5FA                 LD       HL,(ARG1)
E4D9     6E                      LD       L,(HL)
E4DA     26 FF                   LD       H,0FFH
E4DC     22 E5DB                 LD       (KLOC),HL
E4DF     2A E5F8                 LD       HL,(ARG0)
E4E2     5E                      LD       E,(HL)
E4E3     23                      INC      HL
E4E4     56                      LD       D,(HL)
E4E5     EB                      EX       DE,HL
E4E6     7E                      LD       A,(HL)
E4E7     23                      INC      HL
E4E8     D6 02                   SUB      2
E4EA     28 24                   JR       Z,KPUTL3
E4EC     5E                      LD       E,(HL)
E4ED     23                      INC      HL
E4EE     56                      LD       D,(HL)
E4EF     EB                      EX       DE,HL
E4F0     23                      INC      HL
E4F1     23                      INC      HL
E4F2     1F                      RRA
E4F3     47                      LD       B,A
E4F4     C5             KPUTL2:  PUSH     BC
E4F5     ED 4B E5DB              LD       BC,(KLOC)
E4F9     04                      INC      B
E4FA     ED 43 E5DB              LD       (KLOC),BC
E4FE     5E                      LD       E,(HL)
E4FF     23                      INC      HL
E500     56                      LD       D,(HL)
E501     23                      INC      HL
E502     3A E5DA                 LD       A,(KBANK)
E505     E5                      PUSH     HL
E506     2A E5DD                 LD       HL,(KVECT)
E509     CD E451                 CALL     JUMP1
E50C     E1                      POP      HL
E50D     C1                      POP      BC
E50E     10 E4                   DJNZ     KPUTL2
E510     ED 4B E5DB     KPUTL3:  LD       BC,(KLOC)
E514     2A E5DD                 LD       HL,(KVECT)
E517     04             KPUTL4:  INC      B
E518     78                      LD       A,B
E519     FE 28                   CP       40
E51B     30 0F                   JR       NC,KPUTL5
E51D     11 2121                 LD       DE,KSPACE
E520     3A E5DA                 LD       A,(KBANK)
E523     C5                      PUSH     BC
E524     E5                      PUSH     HL
E525     CD E451                 CALL     JUMP1
E528     E1                      POP      HL
E529     C1                      POP      BC
E52A     18 EB                   JR       KPUTL4
E52C     C9             KPUTL5:  RET
                        ;
E52D                    PARM5:   ; for GCMOVE return value
E52D     2A E5F8                 LD       HL,(ARG0)
E530     73                      LD       (HL),E
E531     23                      INC      HL
E532     72                      LD       (HL),D
E533     2A E5FA                 LD       HL,(ARG1)
E536     71                      LD       (HL),C
E537     23                      INC      HL
E538     70                      LD       (HL),B
E539     32 E5E3                 LD       (USRVAL),A
E53C     C9                      RET
                        ;
E53D                    SVOFF:   ; text window offset value save area
E53D     00                      DB       0
                        ;
                        ;;; convart pattern x <--> y ;;;
                        ;
                        ; *** entrys ***
                        ; PROT   : convert pattern by exchange x <--> y
                        ; PROTX  : convert pattern by change   x --> -x
                        ; PROTY  : convert pattern by change   y --> -y
                        ;
                        ; *** paramaters ***
                        ; BUPO1 : in DE  ; this paramater must point to
                        ;                  the first element of array
                        ;                  using in BASIC PUT@,GET@
                        ;                  statement.
                        ;                  this is original pattern and
                        ;                 the value is saved.
                        ; BUPO2 : in BC  ; this paramater is same as
                        ;                  BUPO1 except for that points
                        ;                  to the array prepared for
                        ;                  converted pattern.
                        ;
                        ; *** local variables ***
                        ; PRP : word
                        ;
                        ; !!! caution !!!
```

```
                          ;  (i)  the pattern must be square. so horizon-
                          ;       tal and vertical length must be same
                          ;       value..
                          ;        if used BASIC GET@ statement ,first 2
                          ;       element of array is stored horizontal and
                          ;       vertical length of pattern. this routine
                          ;       obtain from these length from array , so
                          ;       if pattern is made by LET statement you
                          ;       must setup this values in array.
                          ; (ii)  this routine write the converted pat-
                          ;       tern to the array pointed by BUPO2, so
                          ;       this array size must be same or larger
                          ;       than pattern.
                          ;        if not so then may destroy other
                          ;       variable or system program area.
                          ;
E53E    CD E5B6           PROT:    CALL    GETLEN
E541    FD 2A E5DF                 LD      IY,(PRP)
E545    47                         LD      B,A
E546    5F                         LD      E,A
E547    C5                PROT1:   PUSH    BC
E548    2E 80                      LD      L,80H
E54A    FD 22 E5DF        PROT2:   LD      (PRP),IY
E54E    4B                         LD      C,E
E54F    06 08             PROT3:   LD      B,8
E551    DD 66 00                   LD      H,(IX)
E554    DD 23                      INC     IX
E556    7D                PROT4:   LD      A,L
E557    CB 24                      SLA     H
E559    30 05                      JR      NC,PROT5
E55B    FD B6 00                   OR      (IY)
E55E    18 04                      JR      PROT6
E560    2F                PROT5:   CPL
E561    FD A6 00                   AND     (IY)
E564    FD 77 00          PROT6:   LD      (IY),A
E567    FD 19                      ADD     IY,DE
E569    10 EB                      DJNZ    PROT4
E56B    0D                         DEC     C
E56C    20 E1                      JR      NZ,PROT3
E56E    FD 2A E5DF                 LD      IY,(PRP)
E572    CB 3D                      SRL     L
E574    30 D4                      JR      NC,PROT2
E576    FD 23                      INC     IY
E578    C1                         POP     BC
E579    10 CC                      DJNZ    PROT1
E57B    C9                         RET
                          ;
E57C    CD E5B6           PROTX:   CALL    GETLEN
E57F    4B                         LD      C,E
E580    5F                         LD      E,A
E581    19                PROTX1:  ADD     HL,DE
E582    43                         LD      B,E
E583    DD 7E 00          PROTX2:  LD      A,(IX)
E586    DD 23                      INC     IX
E588    C5                         PUSH    BC
E589    06 08                      LD      B,8
E58B    4F                         LD      C,A
E58C    CB 21             PROTX3:  SLA     C
E58E    1F                         RRA
E58F    10 FB                      DJNZ    PROTX3
E591    C1                         POP     BC
E592    2B                         DEC     HL
E593    77                         LD      (HL),A
E594    10 ED                      DJNZ    PROTX2
E596    0D                         DEC     C
E597    C8                         RET     Z
E598    19                         ADD     HL,DE
E599    18 E6                      JR      PROTX1
                          ;
E59B    CD E5B6           PROTY:   CALL    GETLEN
E59E    47                         LD      B,A
E59F    19                PROTY1:  ADD     HL,DE
E5A0    10 FD                      DJNZ    PROTY1
E5A2    4B                         LD      C,E
E5A3    5F                         LD      E,A
E5A4    ED 52             PROTY2:  SBC     HL,DE
E5A6    43                         LD      B,E
E5A7    DD 7E 00          PROTY3:  LD      A,(IX)
E5AA    DD 23                      INC     IX
E5AC    77                         LD      (HL),A
E5AD    23                         INC     HL
E5AE    10 F7                      DJNZ    PROTY3
E5B0    0D                         DEC     C
E5B1    C8                         RET     Z
E5B2    ED 52                      SBC     HL,DE
E5B4    18 EE                      JR      PROTY2
                          ;
E5B6    D5                GETLEN:  PUSH    DE
E5B7    DD E1                      POP     IX
E5B9    AF                         XOR     A
E5BA    60                         LD      H,B
```

```
E5BB     69                        LD      L,C
E5BC     DD 4E 00                  LD      C,(IX)
E5BF     71                        LD      (HL),C
E5C0     23                        INC     HL
E5C1     77                        LD      (HL),A
E5C2     23                        INC     HL
E5C3     DD 46 02                  LD      B,(IX+2)
E5C6     70                        LD      (HL),B
E5C7     23                        INC     HL
E5C8     77                        LD      (HL),A
E5C9     23                        INC     HL
E5CA     22 E5DF                   LD      (PRP),HL
E5CD     11 0004                   LD      DE,4
E5D0     DD 19                     ADD     IX,DE
E5D2     58                        LD      E,B
E5D3     78                        LD      A,B
E5D4     0F                        RRCA
E5D5     0F                        RRCA
E5D6     0F                        RRCA
E5D7     E6 1F                     AND     1FH
E5D9     C9                        RET
                          ;
                          ;
E5DA     00               KBANK:   DEFB    0
E5DB     0000             KLOC:    DEFW    0
E5DD     800A             KVECT:   DEFW    KPUT
                          ;
E5DF     0000             PRP:     DEFW    0
                          ;
E5E1     0000             VARPTR:  DEFW    0
E5E3     00               USRVAL:  DEFB    0
                          ;
                          ;
E5E4                               DEFS    0E5F8H-$
E5F8                      PARAM    EQU     $
E5F8                      ARG0:    DEFS    2
E5FA                      ARG1:    DEFS    2
E5FC                      ARG2:    DEFS    2
E5FE                      ARG3:    DEFS    2
                          ;
                          ;
                          ;
                                   END
```

END ソース・リストの終り

Macros:

Symbols:

ARG0	E5F8	ARG1	E5FA	ARG2	E5FC	ARG3	E5FE
BCOLP	0052	BOOT	E000	BOOT9	E018	CLS	806F
CLSH	8074	CLSS	807A	CRTCC	0051	CRTCP	0050
DSLBNK	E440	ENTRY	E400	EXIT	E434	GCMOV2	829E
GCMOV3	82B5	GCMOV4	82C6	GCMOV5	82C9	GCMOV6	82E9
GCMOV7	82ED	GCMOV8	82F0	GCMOVE	8250	GCREV	823A
GCVAL	838F	GETLEN	E5B6	GHRN	8213	GRBOT	FE80
GRDE	005F	GREB1	005C	GREB2	005D	GREB3	005E
GRST	005C	GRTOP	C000	GRWB	0050	GS	838D
GVRN	815B	GVRN1	817B	GX	8385	GXM	8389
GY	8387	GYM	838B	HBITT	8367	HLINE	82F8
HLINE1	8328	IMAGE	8000	IMAGEB	8000	IMAGEE	8390
JMPTBL	E452	JUMP	E450	JUMP1	E451	KADR	8377
KBANK	E5DA	KBP0	0000	KBP1	0001	KBP8	0008
KBSC1	8275	KBSC2	8276	KBSC3	827C	KBSC4	8282
KBSC5	828C	KBSC6	828D	KBSCAN	8261	KJAHP	00E9
KJALP	00E8	KJDP	00EB	KJEP	00EA	KLOC	E5DB
KPUT	800A	KPUT1	801A	KPUT2	801D	KPUT3	8048
KPUTH	8000	KPUTH1	8009	KPUTL	E4C4	KPUTL1	E4D0
KPUTL2	E4F4	KPUTL3	E510	KPUTL4	E517	KPUTL5	E52C
KPUTLH	E4CD	KSPACE	2121	KVECT	E5DD	LIBMAX	0012
LOAD	E000	PARAM	E5F8	PARM0	E445	PARM1	E49A
PARM2	E4BD	PARM3	E4B7	PARM4	E49F	PARM5	E52D
PARMG	E4C0	PMIXP	0053	PPAL	0054	PROT	E53E
PROT1	E547	PROT2	E54A	PROT3	E54F	PROT4	E556
PROT5	E560	PROT6	E564	PROTX	E57C	PROTX1	E581
PROTX2	E583	PROTX3	E58C	PROTY	E59B	PROTY1	E59F
PROTY2	E5A4	PROTY3	E5A7	PRP	E5DF	PTWOFF	0070
PTWOFI	0078	RLDWH1	8121	RLUPH1	80BA	ROLC	837D
ROLDW	80F4	ROLDW1	8101	ROLDW2	8134	ROLDW3	8149
ROLDWH	810A	ROLLI	817E	ROLLI1	818A	ROLLI2	8199
ROLLI3	81B7	ROLLI4	81B8	ROLLIH	8185	ROLM	837F
ROLN	8381	ROLP	8379	ROLQ	837B	ROLRI	81CC
ROLRI1	81D8	ROLRI2	81E0	ROLRI3	81FE	ROLRI4	81FF
ROLRIH	81D3	ROLUP	808F	ROLUP1	809B	ROLUP2	80CD
ROLUP3	80E2	ROLUPH	80A4	ROLWK	8383	SELBNK	E446
STEP1	0001	STEP2	0008	SVOFF	E53D	USRVAL	E5E3
VARPTR	E5E1	VBITT	836F	VLINE	8334	VLINE1	835E

シンボル・テーブル

索　引

A
AND ……204

B
Bresenhamの方法 ……36
BSAVE ……114

C
CAD ……228
Car ……210
CIRCLE ……52
Cloth ……202
CLS ……115,118
COLOR ……263
COLOR@ ……263
CRT ……13
CRTC ……19,135
CRTディスプレイ ……22

D
DDA ……36,52
Design ……215
DI ……119
dimension ……277

E
EI ……119

G
GET@ ……113,265
Glib ……115
Gエディタ ……108

I
I/Oポート ……118

J
JIS漢字コード ……130

L
LINE ……37,210
LOAD ……222,227
Logic ……228
LP ……31,37,43,67

M
MID. SEL ……88
MON ……132

N
nil ……165

O
OR ……204

P
PAINT ……54
POINT ……31
POINT関数 ……43
PRESET ……30
PSET ……29
PUT@ ……112,265

Q
Qバッファ ……172

R
ROLL ……116,122
Room ……193

S
Sango ……242

SAVE ……222,227
SWAP ……254

U
USR関数 ……115,127

V
VARPTR ……127
VIEW ……59,66
V-RAM ……18

W
WINDOW ……59

X
XOR ……128,167,194
XYプロッタ ……228
X方向反転……116,125

Y
Y方向反転……116,125

ア
アトリビュート……20
アトリビュート・エリア……20
アニメーション ……247,265
アニメーション・ジェネレータ……265
インキング……216
ウィンドウ……57,284
円……45
円グラフ……147
円の方程式……45
オーバフロー……71,78
オリジナル・スクリーン座標系……27
折れ線グラフ……142

カ
ガーベジ・コレクション……168,173
開始角度……53
回転……282,289
回転移動……49,262
拡大・縮小……281,289
カラー・パレット……21
カラー・パレットコントロール・ポート……136
カラー・マップ・テーブル……13,17
漢字ROM ……130,164,191
漢字辞書……170
漢字ワード・プロセッサ……129,164
漢字1行PUT ……129,169
関数……57
逆関数……70
逆転リンク……166
キャラクタ……18,20
キャラクタ座標系……28
キャラクタ・ジェネレータ……19
極座標……47
極座標系……26
切り絵のテクニック……202
グラフィックI/O ……135
グラフィックV-RAM ……20
グラフィック・エディタ……100
グラフィック座標系……27,28
グラフィック・ディスプレイ……13
クリッピング……284,285,291
クロス・カーソル……100,116,127,266
高次関数……65
コサイン……77
弧度……75
コマンド・レジスタ……135

コンピュータ・グラフィックス……3

サ

サイン……75
座標……25,26
サム・チェック……134
三角関数……75
三角形……38,39
四角形……38,42
次元……277
辞書モード……170
指数関数……74
始点座標……37
視野ピラミッド……291
周期……75
周期関数……75
十字カーソル……194,215
終点座標……37
終了角度……53
シンボル・グラフ……156
シンボル・ジェネレータ……97
スクリーン・エディタ……170
スクリーン座標……59
スクリーン座標系……27
スクリーン・レイアウト用紙……205
スクロール……122
スケーリング……281
スタック……243
スタック・ポインタ……243
ストリング・ディスクリプタ……169
スパイラル……51
正弦関数……75
正接関数……78
絶対座標……28
セミカラー・グラフィックス……15
線香花火……256
相対移動量……29
相対座標……28
ソフト・コピー……13

タ

対称形DDA……37
対称軸反転……116,125
対数関数……74
対数グラフ……160
タイル・ストリング……56,88
タイル・パターン・ジェネレータ……93
多角形……38,43
縦・横化……22,23
タンジェント……78
単純形DDA……37
単点透視……293
中間色ジェネレータ……88
直線の方程式……31
直交座標系……26
底……74
ディスプレイ・コントローラ……15
デカルト座標系……26
テキスタイル・デザイン……4
テキストV-RAM……18,19
テキスト・ウィンドウ……119
デジタイザ……193
デジタル微分解析機……36
度……75
透視変換……293
ドット・グラフィックス……18

ハ

ハードウェア……13
ハード・コピー……13,22,146
媒介変数……81
パターン反転……116,125

バック・グラウンド・カラー……30,136
パラメータ……81
パラメータ・レジスタ……135
パレット……114,254
パレット番号……29
バンク……21,289,296
ピクセル……15,17,90
ピッチ……289,296
ビューポート……59,284
フリー・リスト……169
プリンタ……22
フルカラー・グラフィックス……15
フレーム……15
フレーム・バッファ……15
フレーム・バッファ・ディスプレイ……14
プレーン……17
分数関数……69,71
平行移動……279,288
ページ……252
ページング……260
ベクタ・スキャン方式……14
ヘディング……289,296
変態……260
ポインタ……165
放物線……65
放物線のグラフ……65
ボーダー・カラー……135
ポリゴン……242

マ

マジック・リスト……166
マルチ・プレーン……17
無理関数……69
メタモルフォーゼ……260
モニタ……132

ヤ

余弦関数……77

ラ

ライト・ペン……193
ライン・エディタ……172
ライン・スタイル……146
ラジアン……53,75,76
ラスタ・グラフィックス……14
ラスタ・スキャン・ディスプレイ……14
ラスタ・スキャン方式……14
ランダム・スキャン方式……14
リサージュ曲線……83
リスト……164
リスト構造……168,173
リンク……165
ルート関数……69
レイアウト……193
レーダー・チャート……151
ローテーション……230
ロジック回路……228

ワ

ワールド座標……57
ワールド座標系……27
ワイヤ・フレーム……297

1次関数……61
2次関数……65
2次元グラフィックス……277
2次元変換……279
3次元グラフィックス……277
3次元の直交座標系……26
3次元パッケージ……293
3次元変換……288

参 考 文 献

〔1〕 Niklaus Wirth著，片山卓也訳，日本コンピュータ協会発行
「アルゴリズム＋データ構造＝プログラム」
〔2〕 山口富士夫著，日刊工業新聞社発行
「コンピュータディスプレイによる図形処理工学」

プログラム協力

柳楽 直樹

PC-8801 グラフィックスのすべて
アスキー・システム・バンク(PC88＃2)

1982年12月20日 第1版第4刷発行
定価2,500円

著 者 工藤丈彦・榊正憲・田村明史・横溝和宏
発行者 塚本慶一郎
発行所 株式会社アスキー
〒107 港区南青山5－11－5住友青山ビル5F
振 替 東京7-57496
電 話 03－486－7111(代表)

編集担当 井芹昌信
印刷 共同印刷

ISBN4-87148-291-X C3055 ¥2500E

ASCII SYSTEM BANK

PC88編第②巻 内容抜粋

一般的なグラフィック・ディスプレイ
PC-8801の画面に関するハードウェア
座標系と点
直線で図形を描く
円を描く
図形に色を塗る
任意の関数をプロットする
中間色ジェネレータ
タイル・パターン・ジェネレータ
シンボル・ジェネレータ
グラフィック・エディタ
機械語によるツール
折れ線グラフ
円グラフ
レーダー・チャート
シンボル・グラフ
対数グラフ
漢字ワード・プロセッサ
白黒3ページ・モードを使った部屋のレイアウト
タイリングを使った服地のシミュレーション
ロジック回路を描く
パターンの移動
カラー・パレットの利用
ページングの利用
GET@／PUT@の利用
3次元変換
3次元パッケージ
ワイヤ・フレーム

工藤文彦・榊 正憲
田村明史・横溝和宏 共著

アスキー・システム・バンク(PC88#2)

PC-8801 グラフィックスのすべて

定価2,500円

ISBN4-87148-291-X

C3055 ¥2500E